Package: hprimxml (via r-universe)

November 15, 2026

Type Package

Title Écriture et lecture de fichiers HPRIM XML

Version 0.3.0

Description Écriture et lecture de messages HPRIM XML (standard d'interopérabilité français, Interop'Santé) depuis et vers des tables (data.frames, tibbles) : PMSI (RSS MCO), serveur d'actes, états patient. Guidé par les schémas XSD officiels embarqués (versions 1.03a à 2.4). Portage R du package Python pyhprimxml : mêmes tables, mêmes options, sortie XML identique octet pour octet.

License AGPL (>= 3)

Encoding UTF-8

Language fr

Depends R (>= 4.1)

Imports xml2, hms, tibble, tools, openssl, stats, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Collate 'faker_noms.R' 'write.R' 'parse.R' 'xsd.R' 'common.R' 'anonymiser.R' 'depots.R' 'serveur_actes.R' 'etat_patient.R' 'hprimxml-package.R' 'pmsi.R'

URL https://guillaumepressiat.github.io/hprimxml/, https://github.com/GuillaumePressiat/hprimxml

Config/pak/sysreqs libxml2-dev libssl-dev

Repository https://guillaumepressiat.r-universe.dev

Date/Publication 2026-11-15 11:31:53 UTC

RemoteUrl https://github.com/GuillaumePressiat/hprimxml

RemoteRef HEAD

RemoteSha 47886ae914989c8edc269f50ab603fb60f9f656a

Contents

hprim_anonymiser . . . 2
hprim_anonymiser_message . . . 3
hprim_available_versions . . . 4
hprim_decalage_jours . . . 4
hprim_faux_nom . . . 5
HPRIM_NS . . . 5
hprim_parse_xml . . . 6
hprim_pseudonyme . . . 7
hprim_to_xml . . . 7
hprim_validate . . . 9
hprim_write_xml . . . 9
iter_evenements_etat_patient . . . 10
iter_evenements_pmsi . . . 11
iter_evenements_serveur_actes . . . 12
read_evenements_etat_patient . . . 12
read_evenements_pmsi . . . 13
read_evenements_serveur_actes . . . 14
suivi_depots . . . 15
write_evenements_etat_patient . . . 16
write_evenements_pmsi . . . 21
write_evenements_serveur_actes . . . 25

Index **29**

`hprim_anonymiser` *Anonymiser des fichiers HPRIM XML pour des tests locaux*

Description

Chaque fichier est lu, anonymisé et réécrit dans `out_dir` sous un nom pseudonymisé, avec sa propre transformation aléatoire : rien ne relie deux fichiers anonymisés. À graine égale, les fichiers produits sont identiques, octet pour octet, à ceux de `anonymiser()` dans pyhprimxml.

Usage

```
hprim_anonymiser(sources, out_dir, graine = NULL, encoding = "UTF-8")
```

Arguments

`sources`	Fichier(s), dossier ou motif glob.
`out_dir`	Dossier de sortie.
`graine`	Rend le résultat reproductible (même graine, même fichier source -> même fichier anonymisé). Réservé à la doc et aux tests ; sans graine, chaque appel donne un résultat différent.
`encoding`	Encodage des fichiers écrits.

Details

Règles :

- à l'intérieur d'un fichier, tout reste cohérent : une même valeur donne le même remplacement partout dans le fichier ;
- noms et prénoms (patient, professionnels) sont remplacés par des noms français plausibles (faker), le prénom selon le sexe du patient ; `nomNaissance` reste égal à `nomUsuel` quand ils l'étaient ;
- les identifiants (patient, venue, RSS, RUM, actes, intervention, recueil, message), les adresses et les commentaires gardent leur longueur et leur nature de caractères, pour que les fichiers restent conformes au XSD ;
- toutes les dates d'un fichier sont décalées d'un même nombre de jours, tiré au hasard (durées et âges conservés) ; les heures ne changent pas ;
- les codes et les attributs (sexe, mode, action…) ne sont jamais modifiés ;
- le nom de fichier est pseudonymisé comme l'identifiant du message ; un `.ok` présent à côté du fichier source est reproduit.

Ce n'est pas une anonymisation au sens du RGPD : les fichiers restent des données de santé, où codes rares, durées ou unités peuvent permettre des recoupements. Usage : tests locaux.

Value

Les chemins des fichiers anonymisés (invisible).

Examples

```
f <- system.file("extdata", "pmsi_00001.xml", package = "hprimxml")
a <- hprim_anonymiser(f, out_dir = tempfile())
read_evenements_pmsi(a)$evenements[, c("ipp", "nom_usuel", "prenom", "date_naissance")]
```

---

hprim_anonymiser_message

Anonymiser un message lu avec hprim_parse_xml()

Description

Voir `hprim_anonymiser()` pour les règles.

Usage

```
hprim_anonymiser_message(tree, cle, jours = NULL)
```

Arguments

`tree`	Liste renvoyée par `hprim_parse_xml()`.
`cle`	Clé de transformation (propre à un fichier dans `hprim_anonymiser()`).
`jours`	Décalage des dates ; par défaut `hprim_decalage_jours()`.

Value

La liste anonymisée.

`hprim_available_versions` *Versions de schémas HPRIM XML embarquées*

Description

Versions de schémas HPRIM XML embarquées

Usage

```
hprim_available_versions()
```

Value

Vecteur des versions disponibles (`"1.03a"` à `"2.4"`).

Examples

```
hprim_available_versions()
```

`hprim_decalage_jours` *Décalage des dates dérivé de la clé*

Description

Entre -365 et +365 jours, jamais nul ; identique au `decalage_jours()` de pyhprimxml.

Usage

```
hprim_decalage_jours(cle)
```

Arguments

`cle`	Clé secrète.

Value

Nombre de jours.

hprim_faux_nom *Faux nom et faux prénom (faker de l'anonymisation)*

Description

Tirés de listes de noms et prénoms français courants embarquées dans le package (les mêmes que dans pyhprimxml). Une même valeur et une même clé donnent le même remplacement. Le prénom est féminin si `sexe == "F"`, masculin si `"M"`, sinon tiré des deux listes.

Usage

```
hprim_faux_nom(valeur, cle)

hprim_faux_prenom(valeur, cle, sexe = NULL)
```

Arguments

`valeur`	Valeur d'origine.
`cle`	Clé de transformation (propre à un fichier dans `hprim_anonymiser()`).
`sexe`	Sexe du patient (`"F"`, `"M"` ou `NULL`).

Value

Un nom, ou un prénom.

Examples

```
hprim_faux_nom("MARTIN", cle = "k")
hprim_faux_prenom("JEANNE", cle = "k", sexe = "F")
```

HPRIM_NS *Namespace HPRIM XML*

Description

Valeur par défaut de l'argument `xmlns` des fonctions d'écriture.

Usage

```
HPRIM_NS
```

Format

An object of class `character` of length 1.

`hprim_parse_xml` *Lire un message HPRIM XML en liste*

Description

Lecture guidée par le XSD : les éléments répétables sont toujours des listes non nommées (même avec un seul élément), et les valeurs sont typées d'après le XSD (`Date`, `POSIXct` en UTC, `hms`, `integer` ; le reste et les attributs en texte). Le namespace est ignoré. Le résultat est accepté tel quel par `hprim_to_xml()`, et a la même forme que le dictionnaire de `parse_xml()` en Python.

Usage

```
hprim_parse_xml(source, version = NULL, typed = TRUE)
```

Arguments

`source`	Chemin de fichier, chaîne XML ou vecteur raw.
`version`	Version du schéma ; par défaut l'attribut `version` du message.
`typed`	`FALSE` pour garder toutes les valeurs en texte.

Details

Les `dateTime` sont lus en UTC sans décalage : l'heure écrite dans le fichier est conservée, et réécrite à l'identique.

Value

Liste `list(racine = list(...))`.

Examples

```
f <- system.file("extdata", "pmsi_00001.xml", package = "hprimxml")
msg <- hprim_parse_xml(f)
rum <- msg$evenementsPMSI$evenementPMSI[[1]]$rss$rum
length(rum)
rum[[2]]$dateAction
```

hprim_pseudonyme *Pseudonyme déterministe d'une valeur*

Description

Même longueur, même nature de caractères : chiffre -> chiffre, majuscule -> majuscule, minuscule -> minuscule ; ponctuation et espaces conservés ; tout autre caractère -> `X`. Calculé par HMAC-SHA256 avec la clé : identique au `pseudonyme()` de pyhprimxml. C'est la fonction de base de `hprim_anonymiser()`, qui tire une clé aléatoire par fichier.

Usage

```
hprim_pseudonyme(valeur, cle)
```

Arguments

`valeur`	Valeur(s) à pseudonymiser.
`cle`	Clé secrète.

Value

Vecteur de pseudonymes.

Examples

```
hprim_pseudonyme("190101337", cle = "ma-cle")
```

hprim_to_xml *Écrire un message HPRIM XML à partir d'une liste*

Description

La liste suit la même convention que le dictionnaire du package Python pyhprimxml (format xmltodict) :

- une seule clé : l'élément racine ;
- attributs et éléments enfants sont des clés de la même liste nommée (préfixe `@` accepté mais facultatif : le XSD tranche) ;
- un élément répété est une liste non nommée (ou un vecteur atomique) ;
- un élément à contenu simple portant des attributs utilise la clé `#text`.

Usage

```
hprim_to_xml(
  tree,
  version = NULL,
  xmlns = HPRIM_NS,
  encoding = "UTF-8",
  pretty = TRUE,
  drop_empty = TRUE,
  strict = TRUE,
  validate_xsd = FALSE
)
```

Arguments

tree	Liste `list(racine = list(...))`.
version	Version du schéma ; par défaut l'attribut `version` du message, sinon 2.4. Si renseignée, elle remplace l'attribut.
xmlns	`HPRIM_NS` (défaut) ; `NULL` : aucune déclaration ; `""` : écrit `xmlns=""` littéralement sur la racine.
encoding	Encodage déclaré (`"UTF-8"` ou `"ISO-8859-1"`) ; les caractères hors encodage deviennent des références numériques.
pretty	Indentation (deux espaces, mise en forme de libxml2).
drop_empty	Supprime éléments et attributs vides (`NULL`, `NA`, `""`), sauf les éléments complexes obligatoires du XSD.
strict	Erreur si une clé ne correspond à rien dans le XSD (sinon avertissement).
validate_xsd	`FALSE`, `"warn"` ou `"error"` (`TRUE` équivaut à `"error"`).

Details

L'ordre des clés n'a pas d'importance : les enfants sont émis dans l'ordre du XSD. Le document est construit et mis en forme par xml2 (libxml2), comme lxml en Python : la sortie est identique, octet pour octet, à celle de `to_xml()`.

Value

Le message XML (chaîne UTF-8), déclaration comprise.

Examples

```
f <- system.file("extdata", "pmsi_00001.xml", package = "hprimxml")
msg <- hprim_parse_xml(f)
cat(substr(hprim_to_xml(msg, xmlns = NULL), 1, 200))
```

hprim_validate *Valider un message HPRIM XML contre son XSD*

Description

La variante de schéma (avec ou sans namespace) est choisie d'après le message lui-même.

Usage

```
hprim_validate(x, version = NULL)
```

Arguments

x	Message : chaîne XML, chemin de fichier, vecteur raw ou document xml2.
version	Version du schéma ; par défaut l'attribut `version` du message.

Value

Vecteur des erreurs, vide si le message est conforme. Si le XSD officiel de la version est lui-même invalide (cas du XSD PMSI 1.06), une erreur de classe `hprim_schema_error` est levée : la validation est impossible avec cette version.

Examples

```
f <- system.file("extdata", "pmsi_00001.xml", package = "hprimxml")
hprim_validate(f)
```

hprim_write_xml *Écrire un message HPRIM XML dans un fichier*

Description

Écrire un message HPRIM XML dans un fichier

Usage

```
hprim_write_xml(tree, path, encoding = "UTF-8", ...)
```

Arguments

tree	Liste `list(racine = list(...))`.
path	Fichier de sortie.
encoding	Encodage du fichier.
...	Autres options de `hprim_to_xml()`.

Value

Le chemin écrit (invisible).

`iter_evenements_etat_patient` *Arbres des messages evenementsServeurEtatsPatient, un par événement*

Description

Arbres des messages evenementsServeurEtatsPatient, un par événement

Usage

```
iter_evenements_etat_patient(
  evenements,
  diagnostics = NULL,
  entete = NULL,
  version = NULL,
  colmap = NULL,
  par_diagnostic = FALSE
)
```

Arguments

`evenements`, `diagnostics`	Tables.
`entete`	Liste : `emetteur`, `destinataire`, et en option `version`, `mode_traitement`, `acquittement_attendu`, `commentaire`, `date_heure_production`, `emetteur_categorie`, `destinataire_categorie`. Chaque clé peut aussi être une colonne de `evenements` (prioritaire).
`version`	Version du schéma ; sinon `entete$version` ou colonne `version`, sinon 2.4.
`colmap`	Renommage de colonnes `list(ancien = "nouveau")`.
`par_diagnostic`	Un message par diagnostic (unité atomique) au lieu d'un par événement. Le contexte (patient, venue, repère) est répété ; `evt_id` vaut `{evt_id}_{rang}`, et le gabarit `filename` dispose aussi de `{evt_origine}` et `{rang}`.

`iter_evenements_pmsi` *Arbres (format liste) des messages evenementsPMSI, un par événement ou par RUM*

Description

Arbres (format liste) des messages evenementsPMSI, un par événement ou par RUM

Usage

```
iter_evenements_pmsi(
  evenements,
  rum,
  diagnostics = NULL,
  actes_ccam = NULL,
  entete = NULL,
  version = NULL,
  colmap = NULL,
  par_rum = FALSE
)
```

Arguments

`evenements`, `rum`, `diagnostics`, `actes_ccam`	Tables décrites ci-dessus.
`entete`	Liste : `emetteur`, `destinataire`, et en option `version`, `mode_traitement`, `acquittement_attendu`, `commentaire`, `date_heure_production`, `emetteur_categorie`, `destinataire_categorie`. Chaque clé peut aussi être une colonne de `evenements` (prioritaire).
`version`	Version du schéma ; sinon `entete$version` ou colonne `version`, sinon 2.4.
`colmap`	Renommage de colonnes `list(ancien = "nouveau")`.
`par_rum`	Un fichier par RUM ; `evt_id` vaut alors `{evt_id}_{rum_id}` et le gabarit dispose aussi de `{rss_id}` et `{rum_id}`.

Value

Liste d'éléments `list(evt = ligne, tree = arbre)`.

Examples

```
evenements <- data.frame(evt_id = "S1", ipp = "123", sexe = "F", date_naissance = as.Date("1960-01-01"),
                         nda = "240000001", venue_date_entree = as.Date("2024-05-02"))
rum <- data.frame(evt_id = "S1", rum_id = 1L, um = "1201", um_date_entree = as.Date("2024-05-02"),
                  um_mode_entree = "8", dp = "I21.0")
arbres <- iter_evenements_pmsi(evenements, rum, entete = list(emetteur = "DIM", destinataire = "DPI"))
str(arbres[[1]]$tree$evenementsPMSI$evenementPMSI$rss$rum[[1]], max.level = 1)
cat(hprim_to_xml(arbres[[1]]$tree, xmlns = NULL))
```

iter_evenements_serveur_actes

Arbres des messages evenementsServeurActes, un par événement

Description

Arbres des messages evenementsServeurActes, un par événement

Usage

```
iter_evenements_serveur_actes(
  evenements,
  actes_ccam = NULL,
  actes_ngap = NULL,
  entete = NULL,
  version = NULL,
  colmap = NULL,
  par_acte = FALSE
)
```

Arguments

evenements, actes_ccam, actes_ngap	Tables (actes_ccam : CCAM ; actes_ngap : NGAP).
entete	Liste : emetteur, destinataire, et en option version, mode_traitement, acquittement_attendu, commentaire, date_heure_production, emetteur_categorie, destinataire_categorie. Chaque clé peut aussi être une colonne de evenements (prioritaire).
version	Version du schéma ; sinon entete$version ou colonne version, sinon 2.4.
colmap	Renommage de colonnes list(ancien = "nouveau").
par_acte	Un message par acte (unité atomique) au lieu d'un par événement. Le contexte (patient, venue, intervention) est répété ; evt_id vaut {evt_id}_{rang}, et le gabarit filename dispose aussi de {evt_origine}, {rang} et {acte_id}.

read_evenements_etat_patient

Lire des messages evenementsServeurEtatsPatient vers des tables

Description

Lire des messages evenementsServeurEtatsPatient vers des tables

Usage

```
read_evenements_etat_patient(
  sources,
  extra = NULL,
  version = NULL,
  exiger_ok = FALSE
)
```

Arguments

sources	Fichier(s), dossier, motif glob ou vecteur raw.
extra	Colonnes-chemins supplémentaires, par table : `list(evenements = "patient/@anonyme", rum = "@classant")`.
version	Forcer une version de schéma.
exiger_ok	Ne lire que les `.xml` accompagnés de leur `.ok`.

Value

Liste de tibbles `evenements`, `diagnostics`.

Examples

```
f <- system.file("extdata", "etat_patient_00001.xml", package = "hprimxml")
t <- read_evenements_etat_patient(f)
t$diagnostics

# un IGS2 se relit par des colonnes-chemins (voir l'exemple de write_evenements_etat_patient)
# read_evenements_etat_patient("sortie/igs2", extra = list(evenements = c("igs2/@age", "igs2/valeurGlobale")))
```

---

read_evenements_pmsi *Lire des messages evenementsPMSI (RSS MCO) vers des tables*

Description

Inverse de `write_evenements_pmsi()` : mêmes tables, mêmes colonnes. `evt_id` est le nom du fichier sans extension ; `rum_id` le rang du RUM dans le RSS.

Usage

```
read_evenements_pmsi(
  sources,
  extra = NULL,
  version = NULL,
  exiger_ok = FALSE,
  format_diagnostics = c("long", "colonnes")
)
```

Arguments

sources	Fichier(s), dossier, motif glob ou vecteur raw.
extra	Colonnes-chemins supplémentaires, par table : `list(evenements = "patient/@anonyme", rum = "@classant")`.
version	Forcer une version de schéma.
exiger_ok	Ne lire que les `.xml` accompagnés de leur `.ok`.
format_diagnostics	`"long"` (table `diagnostics`) ou `"colonnes"` (`dp`, `dr`, `das`, `dad` dans `rum`, la table `diagnostics` étant alors vide).

Value

Liste de tibbles `evenements`, `rum`, `diagnostics`, `actes_ccam`.

Examples

```
f <- system.file("extdata", "pmsi_00001.xml", package = "hprimxml")
t <- read_evenements_pmsi(f)
t$rum[, c("evt_id", "rum_id", "rum_emetteur", "um", "um_date_entree")]
t$diagnostics

# diagnostics en colonnes de `rum`, et un attribut sans colonne dédiée
t <- read_evenements_pmsi(f, format_diagnostics = "colonnes", extra = list(rum = "@classant"))
t$rum[, c("rum_id", "dp", "dr", "das", "@classant")]
```

---

read_evenements_serveur_actes

Lire des messages evenementsServeurActes vers des tables

Description

Lire des messages evenementsServeurActes vers des tables

Usage

```
read_evenements_serveur_actes(
  sources,
  extra = NULL,
  version = NULL,
  exiger_ok = FALSE
)
```

Arguments

sources	Fichier(s), dossier, motif glob ou vecteur raw.
extra	Colonnes-chemins supplémentaires, par table : `list(evenements = "patient/@anonyme", rum = "@classant")`.
version	Forcer une version de schéma.
exiger_ok	Ne lire que les `.xml` accompagnés de leur `.ok`.

Value

Liste de tibbles `evenements`, `actes_ccam` (CCAM), `actes_ngap` (NGAP, avec une colonne liste `nabm`).

Examples

```
f <- system.file("extdata", "actes_00001.xml", package = "hprimxml")
t <- read_evenements_serveur_actes(f)
t$evenements[, c("evt_id", "ipp", "nda", "intervention_id")]
t$actes_ccam[, c("evt_id", "acte_id", "code", "date_execution", "medecin")]
```

---

suivi_depots *État des dépôts : quels .xml ont leur marqueur .ok*

Description

Protocole : l'émetteur dépose le `.xml`, puis, une fois celui-ci écrit intégralement, un `.ok` vide de même nom (`fichier_ok = TRUE` des fonctions d'écriture). Le destinataire ne traite un `.xml` qu'une fois son `.ok` présent (`exiger_ok = TRUE` des fonctions de lecture).

Usage

```
suivi_depots(fichiers, extension_ok = ".ok")
```

Arguments

fichiers	Dossier, motif glob ou vecteur de `.xml`.
extension_ok	Extension des marqueurs.

Value

Tibble : `evt_id`, `fichier`, `date_xml`, `complet`, `date_ok`. Les `.ok` sans `.xml` correspondant sont ajoutés avec `fichier` manquant et `complet` faux.

Examples

```
ancien <- setwd(tempdir())
evenements <- data.frame(evt_id = "S1", ipp = "123", sexe = "F", date_naissance = as.Date("1960-01-01"),
                         nda = "240000001", venue_date_entree = as.Date("2024-05-02"))
rum <- data.frame(evt_id = "S1", rum_id = 1:2, um = c("1201", "3405"),
            um_date_entree = as.Date(c("2024-05-02", "2024-05-04")), um_mode_entree = c("8", "6"),
                  dp = c("I21.0", "I50.1"))
write_evenements_pmsi(evenements, rum, entete = list(emetteur = "DIM", destinataire = "DPI"),
                      out_dir = "depot", par_rum = TRUE, fichier_ok = TRUE)
file.remove("depot/S1_2.ok")            # dépôt « en cours »
suivi_depots("depot")[, c("evt_id", "complet")]
read_evenements_pmsi("depot", exiger_ok = TRUE)$evenements$evt_id
setwd(ancien)
```

---

write_evenements_etat_patient

Écrire des messages evenementsServeurEtatsPatient depuis des tables

---

Description

Un fichier par événement. `evenements` : patient, venue, intervention, et le repère : `date_observation`, `heure_observation` (élément `dateObservation`), ou `recueil_id`, `recueil_date`, `recueil_heure`, `recueil_uf` (élément `recueil` : date et UF où les états ont été observés, le repère le plus précis) ; les colonnes-chemins sont relatives à la racine du message (ex. `igs2/@age`). `diagnostics` : une ligne par diagnostic, `evt_id`, `type` (dp, dr, da, ds, dd, fpc, mm, ae), `code`, et en option `date_action`, `action`.

Usage

```
write_evenements_etat_patient(
  evenements,
  diagnostics = NULL,
  entete = NULL,
  out_dir,
  filename = "{evt_id}.xml",
  version = NULL,
  xmlns = HPRIM_NS,
  encoding = "UTF-8",
  validate_xsd = "error",
  colmap = NULL,
  identifiant_egal_nom_fichier = TRUE,
  fichier_ok = FALSE,
  par_diagnostic = FALSE
)
```

Arguments

evenements, diagnostics	Tables.
entete	Liste : emetteur, destinataire, et en option version, mode_traitement, acquittement_attendu, commentaire, date_heure_production, emetteur_categorie, destinataire_categorie. Chaque clé peut aussi être une colonne de evenements (prioritaire).
out_dir	Dossier de sortie.
filename	Gabarit ("{evt_id}.xml") ou fonction ligne -> nom.
version	Version du schéma ; sinon entete$version ou colonne version, sinon 2.4.
xmlns	HPRIM_NS (défaut), NULL (pas de namespace) ou "" (xmlns="").
encoding	Encodage des fichiers.
validate_xsd	"error" (défaut), "warn" ou FALSE.
colmap	Renommage de colonnes list(ancien = "nouveau").
identifiant_egal_nom_fichier	L'identifiantMessage est le nom du fichier sans extension (défaut) ; une colonne identifiant_message contradictoire est une erreur.
fichier_ok	Dépose un .ok vide une fois chaque .xml écrit intégralement.
par_diagnostic	Un message par diagnostic (unité atomique) au lieu d'un par événement. Le contexte (patient, venue, repère) est répété ; evt_id vaut {evt_id}_{rang}, et le gabarit filename dispose aussi de {evt_origine} et {rang}.

Details

Le XSD impose un repère (intervention, dateObservation ou recueil) et un contenu exclusif : diagnostics, ou IGS2, ou dépendances… En 2.4, un élément <divers/> est émis.

Unité atomique des états patient (pour les diagnostics) : le diagnostic, porteur de l'attribut action et d'un identifiant. par_diagnostic = TRUE écrit un message par diagnostic. L'IGS2, unique par message, est atomique par construction.

Value

Les chemins écrits (invisible).

Examples

```
# Les exemples écrivent dans un dossier temporaire
ancien <- setwd(tempdir())
# IGS2 détaillé dans un message états patient (evenementsServeurEtatsPatient).
# Des mesures brutes (pires valeurs des 24 premières heures), on déduit pour chaque item
# la classe attendue par le XSD (attribut de <igs2>) et ses points, puis le score global.
# Un fichier par patient : l'élément <igs2> n'est pas répétable dans un message.
library(hprimxml)
library(tibble)

ts <- function(x) as.POSIXct(x, tz = "UTC")
```

```
entete <- list(
  version = "2.00",                    # version HPRIM XML des messages
  emetteur = "REA",
  destinataire = "DIM",
  date_heure_production = ts("2024-03-06 09:00:00")
)

# Pires valeurs des 24 premières heures, une ligne par patient
mesures <- tibble(
  evt_id = c("IGS240001", "IGS240002"),
  age = c(67, 82),
  frequence_cardiaque = c(128, 105),    # /min
  pas = c(85, 210),                     # pression artérielle systolique, mmHg
  temperature = c(38.2, 39.4),          # °C
  ventile = c(TRUE, FALSE),             # ventilation mécanique ou CPAP
  pao2_fio2 = c(180, NA),               # mmHg, si ventilé
  diurese = c(0.8, 1.6),                # L/24 h
  uree = c(18, 8),                      # mmol/L
  leucocytes = c(14, 22),               # 10³/mm³
  kaliemie = c(4.1, 5.4),               # mmol/L
  natremie = c(132, 147),               # mmol/L
  hco3 = c(17, 22),                     # mmol/L
  bilirubine = c(25, 110),              # µmol/L
  ictere = c(FALSE, TRUE),              # la bilirubine ne compte que si ictère
  glasgow = c(13, 15),
  maladie_chronique = c("aucune", "cancerMetastatique"),
  mode_admission = c("medecine", "chirUrgente")
)

# Barème IGS2 : attribut XSD -> mesure, bornes, points par classe XSD
# Source : grille et définitions de la SFAR, https://sfar.org/scores/igs2.php
# (Le Gall JR et coll., JAMA 1993; 270: 2957-2963). Recueil sur les 24 heures suivant
# l'admission : valeur la plus défavorable de chaque item ; Glasgow estimé avant sédation ;
# diurèse ramenée à 24 h si le séjour est plus court ; en cas d'arrêt cardiaque, FC 0,
# PAS 0 et Glasgow < 6. Unités : urée en mmol/L (seuils SFAR en g/L : 0,6 et 1,8),
# leucocytes en 10³/mm³, bilirubine en µmol/L (seuils en mg/L : 40 et 60).
# Les bornes sont fermées à gauche : 40 tombe dans « 40A59 ».
BAREME <- list(
  age = list("age", c(40, 60, 70, 75, 80),
             c(inferieurA40 = 0, `40A59` = 7, `60A69` = 12, `70A74` = 15, `75A79` = 16,
               superieurEgalA80 = 18)),
  frequenceCardiaque = list("frequence_cardiaque", c(40, 70, 120, 160),
         c(inferieurA40 = 11, `40A69` = 2, `70A119` = 0, `120A159` = 4, superieurEgalA160 = 7)),
  tensionArterielleSystolique = list("pas", c(70, 100, 200),
             c(inferieurA70 = 13, `70A99` = 5, `100A199` = 0, superieurEgalA200 = 2)),
  temperature = list("temperature", 39, c(inferieurA39 = 0, superieurEgalA39 = 3)),
 paO2_FIO2 = list("pao2_fio2", c(100, 200), c(inferieurA100 = 11, `100A199` = 9, superieurEgalA200 = 6)),
 diurese = list("diurese", c(0.5, 1), c(`inferieurA0,5` = 11, `0,5A0,999` = 4, superieurA1 = 0)),
 ureeSanguine = list("uree", c(10, 30), c(inferieurA10 = 0, `10A29,9` = 6, superieurA30 = 10)),
 leucocytes = list("leucocytes", c(1, 20), c(inferieurA1 = 12, `1A19,9` = 0, superieurEgalA20 = 3)),
 kaliemie = list("kaliemie", c(3, 5), c(inferieurA3 = 3, `3A4,9` = 0, superieurEgalA5 = 3)),
 natremie = list("natremie", c(125, 145), c(inferieurA125 = 5, `125A144` = 0, superieurEgalA145 = 1)),
```

```
  HCO3 = list("hco3", c(15, 20), c(inferieurA15 = 6, `15A19` = 3, superieurEgalA20 = 0)),
  bilirubine = list("bilirubine", c(68.4, 102.6),
            c(`inferieurA68,4` = 0, `68,4A102,5` = 4, `superieurA102,6` = 9)),
  scoreGlasgow = list("glasgow", c(6, 9, 11, 14),
            c(inferieurA6 = 26, `6A8` = 13, `9A10` = 7, `11A13` = 5, `14A18` = 0))
)
POINTS_QUALITATIFS <- list(
  maladieChronique = list("maladie_chronique",
         c(aucune = 0, cancerMetastatique = 9, maladieHematologiqueMaligne = 10, sida = 17)),
 modeAdmission = list("mode_admission", c(chirProgrammee = 0, medecine = 6, chirUrgente = 8))
)

# Classe d'une mesure : libelles[i] si bornes[i-1] <= mesure < bornes[i]
classer <- function(x, bornes, libelles) {
  as.character(cut(x, c(-Inf, bornes, Inf), labels = libelles, right = FALSE))
}

igs2 <- mesures
for (attr in names(BAREME)) {
  b <- BAREME[[attr]]
  classe <- classer(mesures[[b[[1]]]], b[[2]], names(b[[3]]))
 if (attr == "paO2_FIO2") classe <- ifelse(mesures$ventile, classe, "rien")  # ni ventilé ni CPAP
 if (attr == "bilirubine") classe <- ifelse(mesures$ictere, classe, "inferieurA68,4")  # non ictérique
  igs2[[attr]] <- classe
  igs2[[paste0("points_", attr)]] <- as.integer(c(b[[3]], rien = 0)[classe])
}
for (attr in names(POINTS_QUALITATIFS)) {
  q <- POINTS_QUALITATIFS[[attr]]
  igs2[[attr]] <- mesures[[q[[1]]]]
  igs2[[paste0("points_", attr)]] <- as.integer(q[[2]][mesures[[q[[1]]]]])
}
igs2$valeurGlobale <- as.integer(rowSums(igs2[grep("^points_", names(igs2))]))
attributs <- c(names(BAREME), names(POINTS_QUALITATIFS))

# Patient et venue, une ligne par message
evenements <- tibble(
  evt_id = c("IGS240001", "IGS240002"),
  ipp = c("190101337", "190101338"), sexe = c("M", "F"),
  date_naissance = as.Date(c("1956-05-14", "1941-11-02")),
  nda = c("240000123", "240000124"), venue_date_entree = as.Date("2024-03-05"),
  # repère du message : le recueil, avec sa date et l'UF de réanimation où les états ont
  # été observés (le récepteur retrouve ainsi le RUM, même un jour de mutation)
  recueil_id = c("REC240001", "REC240002"),
  recueil_date = as.Date("2024-03-06"),
  recueil_uf = "4501"
)

# L'IGS2 passe par des colonnes-chemins relatives à la racine du message
evenements$`igs2/@action` <- "creation"
for (a in attributs) evenements[[paste0("igs2/@", a)]] <- igs2[[a]]
evenements$`igs2/dateAction` <- ts("2024-03-06 08:00:00")
evenements$`igs2/valeurGlobale` <- igs2$valeurGlobale
```

```
fichiers <- write_evenements_etat_patient(
  evenements, NULL,                    # pas de table de diagnostics ici
  entete = entete,
  out_dir = "sortie/igs2",             # -> sortie/igs2/IGS240001.xml, IGS240002.xml
  xmlns = NULL
)

# Contrôle : une ligne par item, avec la classe et les points de chaque patient
controle <- tibble(item = attributs)
for (i in seq_len(nrow(igs2))) {
  controle[[paste0("classe_", igs2$evt_id[i])]] <- unlist(igs2[i, attributs])
}
for (i in seq_len(nrow(igs2))) {
 controle[[paste0("points_", igs2$evt_id[i])]] <- unlist(igs2[i, paste0("points_", attributs)])
}
# Régénère data/xml_tests/etat_patient/00001.xml : diagnostics DP et DR.
library(hprimxml)
library(tibble)

ts <- function(x) as.POSIXct(x, tz = "UTC")

entete <- list(
  version = "2.00",                    # version HPRIM XML des messages
  emetteur = "TEST",
  destinataire = NULL,
  acquittement_attendu = "non",
  commentaire = "HPRIM-TEST",
  date_heure_production = ts("2024-01-01 01:01:01")
)

# Une ligne par événement : patient, venue, intervention
# (ici, les colonnes-chemins sont relatives à la racine du message)
evenements <- tibble(
  evt_id = "00001_esep",
  ipp = "123456789", ipp_recepteur = "123456789",
  sexe = "M", nom_usuel = "SANDWICK", prenom = "JOHN",
  date_naissance = as.Date("1970-01-01"),
  nda = "987654321", nda_recepteur = "987654321",
 venue_date_entree = as.Date("2024-01-01"), venue_heure_entree = hms::as_hms("08:00:00"),
  venue_uf_responsable = "4321",
  intervention_id = "000000000001", intervention_uf = "4321",
  `patient/@confidentiel` = "non", `venue/@prive` = "non",
  `patient/identifiant/emetteur/@etat` = "permanent",
  `patient/identifiant/emetteur/@portee` = "local",
  `patient/identifiant/emetteur/@referent` = "non",
  `patient/identifiant/recepteur/@etat` = "permanent",
  `patient/identifiant/recepteur/@portee` = "local",
  `patient/identifiant/recepteur/@referent` = "non"
)

# Une ligne par diagnostic ; type dans dp, dr, da, ds, dd, fpc, mm, ae
diagnostics <- tibble(
  evt_id = "00001_esep",
```

```
  type = c("dp", "dr"),
  code = c("Z511", "C10"),
  date_action = ts("2024-01-01 08:10:00")
)

fichiers <- write_evenements_etat_patient(
  evenements, diagnostics,
  entete = entete,
  out_dir = "sortie/etat_patient",      # -> sortie/etat_patient/00001_esep.xml
  validate_xsd = "warn"
)
# IGS2 : classe et points de chaque item, par patient
print(controle, width = Inf)
igs2$valeurGlobale
setwd(ancien)
```

`write_evenements_pmsi` *Écrire des messages evenementsPMSI (RSS MCO) depuis des tables*

Description

Un fichier par séjour, ou par RUM avec `par_rum = TRUE`. **Unité atomique du PMSI : le RUM**, seul porteur de l'attribut `action` et d'un identifiant. Mêmes tables, mêmes colonnes et même sortie (octet pour octet) que `write_evenements_pmsi()` du package Python pyhprimxml.

Usage

```
write_evenements_pmsi(
  evenements,
  rum,
  diagnostics = NULL,
  actes_ccam = NULL,
  entete = NULL,
  out_dir,
  filename = "{evt_id}.xml",
  version = NULL,
  xmlns = HPRIM_NS,
  encoding = "UTF-8",
  validate_xsd = "error",
  colmap = NULL,
  par_rum = FALSE,
  identifiant_egal_nom_fichier = TRUE,
  fichier_ok = FALSE,
  actes = NULL
)
```

Arguments

`evenements`, `rum`, `diagnostics`, `actes_ccam`
: Tables décrites ci-dessus.

`entete`
: Liste : `emetteur`, `destinataire`, et en option `version`, `mode_traitement`, `acquittement_attendu`, `commentaire`, `date_heure_production`, `emetteur_categorie`, `destinataire_categorie`. Chaque clé peut aussi être une colonne de `evenements` (prioritaire).

`out_dir`
: Dossier de sortie.

`filename`
: Gabarit (`"{evt_id}.xml"`) ou fonction ligne -> nom.

`version`
: Version du schéma ; sinon `entete$version` ou colonne `version`, sinon 2.4.

`xmlns`
: `HPRIM_NS` (défaut), `NULL` (pas de namespace) ou `""` (`xmlns=""`).

`encoding`
: Encodage des fichiers.

`validate_xsd`
: `"error"` (défaut), `"warn"` ou `FALSE`.

`colmap`
: Renommage de colonnes `list(ancien = "nouveau")`.

`par_rum`
: Un fichier par RUM ; `evt_id` vaut alors `{evt_id}_{rum_id}` et le gabarit dispose aussi de `{rss_id}` et `{rum_id}`.

`identifiant_egal_nom_fichier`
: L'identifiantMessage est le nom du fichier sans extension (défaut) ; une colonne `identifiant_message` contradictoire est une erreur.

`fichier_ok`
: Dépose un `.ok` vide une fois chaque `.xml` écrit intégralement.

`actes`
: Ancien nom de `actes_ccam`, encore accepté avec un avertissement.

Details

Tables (data.frame ou tibble) :

- `evenements` : une ligne par séjour. Obligatoires : `evt_id`, `ipp`, `sexe`, `date_naissance`, `nda`, `venue_date_entree`.
- `rum` : une ligne par RUM. Obligatoires : `evt_id`, `rum_id`, `um`, `um_date_entree`, `um_mode_entree`. Diagnostics possibles en colonnes `dp`, `dr`, `das`, `dad` (`das` et `dad` : colonne liste ou chaîne séparée par des virgules).
- `diagnostics` : format long, `evt_id`, `rum_id`, `type` (DP, DR, DAS, DAD), `code`.
- `actes_ccam` : une ligne par acte CCAM, `evt_id`, `rum_id`, `code`, `phase`, `activite`.

Toute colonne dont le nom contient `/` ou commence par `@` est une colonne-chemin, relative au noeud que décrit la table (ex. `patient/@anonyme`, `@classant`).

Value

Les chemins écrits (invisible).

Examples

```
# Les exemples écrivent dans un dossier temporaire
ancien <- setwd(tempdir())
# Séjour avec mutation : entrée par les urgences en cardiologie (UM 1201), puis mutation
# en médecine (UM 3405), sortie au domicile. Un RSS de deux RUM, écrit sans namespace,
# d'abord en un fichier par séjour, puis en RUM unitaires (un fichier par RUM).
library(hprimxml)
library(tibble)

ts <- function(x) as.POSIXct(x, tz = "UTC")

entete <- list(
  version = "2.00",                     # version HPRIM XML des messages
  emetteur = "DIM",
  destinataire = "DPI",
  date_heure_production = ts("2024-03-10 08:00:00")   # par défaut : maintenant
)

# Une ligne par séjour
evenements <- tibble(
  evt_id = "S240001",                   # -> S240001.xml, identifiantMessage S240001
  ipp = "190101337",
  sexe = "F",
  date_naissance = as.Date("1958-03-12"),
  nda = "240000123",
  venue_date_entree = as.Date("2024-03-04"),
  venue_heure_entree = hms::as_hms("10:20:00"),
  venue_date_sortie = as.Date("2024-03-09"),
  rss_emetteur = "240000123"
)

# Une ligne par RUM
rum <- tibble(
  evt_id = "S240001",
  rum_id = 1:2,
  rum_emetteur = c("S240001_1", "S240001_2"),   # identifiant XML du RUM
  um = c("1201", "3405"),
  um_date_entree = as.Date(c("2024-03-04", "2024-03-06")),
  um_heure_entree = hms::as_hms(c("10:20:00", "14:00:00")),
  um_mode_entree = c("8", "6"),         # domicile, puis mutation
  um_provenance = c("5", "1"),          # urgences, puis MCO
  um_date_sortie = as.Date(c("2024-03-06", "2024-03-09")),
  um_mode_sortie = c("6", "8"),         # mutation, puis domicile
  um_destination = c("1", NA),          # vers MCO
  `medecinResponsable/noRPPS` = c("10003456789", "10009876543")
)

# Une ligne par diagnostic (le point des codes CIM-10 est retiré à l'écriture)
diagnostics <- tibble(
  evt_id = "S240001",
  rum_id = c(1L, 1L, 1L, 1L, 2L, 2L, 2L),
  type = c("DP", "DAS", "DAS", "DAS", "DP", "DAS", "DAS"),
```

```
  code = c("I21.0", "E11.9", "I10", "E78.0", "I50.1", "N18.3", "E11.9")
)

# Une ligne par acte CCAM
actes_ccam <- tibble(
  evt_id = "S240001",
  rum_id = c(1L, 1L, 2L),
  code = c("DEQP003", "ZBQK002", "DEQP003"),
  phase = 0L,
  activite = 1L,
 date_realisation = ts(c("2024-03-04 10:45:00", "2024-03-04 11:30:00", "2024-03-06 15:00:00"))
)

# Un fichier par séjour : sortie/sejour/S240001.xml
write_evenements_pmsi(
  evenements, rum, diagnostics, actes_ccam,
  entete = entete, out_dir = "sortie/sejour", xmlns = NULL
)

# RUM unitaires, avec marqueur de dépôt complet :
# sortie/rum/S240001_1.xml puis S240001_1.ok, sortie/rum/S240001_2.xml puis S240001_2.ok
write_evenements_pmsi(
  evenements, rum, diagnostics, actes_ccam,
  entete = entete, out_dir = "sortie/rum", xmlns = NULL, par_rum = TRUE, fichier_ok = TRUE
)

# Relecture : les mêmes tables (ici, diagnostics en colonnes de `rum`)
relu <- read_evenements_pmsi("sortie/sejour", format_diagnostics = "colonnes")
rum_relu <- relu$rum[, c("evt_id", "rum_id", "rum_emetteur", "um", "dp", "das")]
# Trois façons équivalentes de fournir les diagnostics d'un RUM avec plusieurs DAS.
# Les trois écritures produisent exactement le même fichier XML.
library(hprimxml)
library(tibble)

entete <- list(
  version = "2.00",                     # version HPRIM XML des messages
  emetteur = "DIM",
  destinataire = "DPI",
  date_heure_production = as.POSIXct("2024-03-10 08:00:00", tz = "UTC")
)

evenements <- tibble(
  evt_id = "S240002", ipp = "190101338", sexe = "M",
  date_naissance = as.Date("1951-07-02"), nda = "240000124",
  venue_date_entree = as.Date("2024-03-04"), rss_emetteur = "240000124"
)

rum <- tibble(
  evt_id = "S240002", rum_id = 1L, um = "1201",
  um_date_entree = as.Date("2024-03-04"), um_mode_entree = "8"
)

# 1. Format long : une table `diagnostics`, une ligne par code
```

```
diagnostics <- tibble(
  evt_id = "S240002",
  rum_id = 1L,
  type = c("DP", "DAS", "DAS", "DAS", "DAS"),
  code = c("I21.0", "E11.9", "I10", "E78.0", "N18.3")
)
write_evenements_pmsi(evenements, rum, diagnostics, entete = entete, out_dir = "sortie/long")

# 2. Colonnes dans `rum`, DAS en colonne liste
rum_liste <- rum
rum_liste$dp <- "I21.0"
rum_liste$das <- list(c("E11.9", "I10", "E78.0", "N18.3"))
write_evenements_pmsi(evenements, rum_liste, entete = entete, out_dir = "sortie/liste")

# 3. Colonnes dans `rum`, DAS en chaîne séparée par des virgules
rum_chaine <- rum
rum_chaine$dp <- "I21.0"
rum_chaine$das <- "E11.9, I10, E78.0, N18.3"
write_evenements_pmsi(evenements, rum_chaine, entete = entete, out_dir = "sortie/chaine")

# Une chaîne se convertit aussi en liste d'une ligne : strsplit(rum_chaine$das, ", ")
fichiers <- lapply(c("long", "liste", "chaine"), function(f)
  readBin(file.path("sortie", f, "S240002.xml"), "raw", 1e6))
stopifnot(identical(fichiers[[1]], fichiers[[2]]), identical(fichiers[[1]], fichiers[[3]]))
list.files("sortie", recursive = TRUE)
rum_relu
setwd(ancien)
```

`write_evenements_serveur_actes` *Écrire des messages evenementsServeurActes (actes CCAM) depuis des tables*

Description

Un fichier par événement. `evenements` : patient, venue (nda, `venue_*`), intervention (`intervention_*`, `demande_*`). `actes_ccam` : une ligne par acte CCAM, avec `evt_id`, `acte_id`, `code`, `activite`, `phase`, `date_execution`, `medecin` et en option `heure_execution`, `uf`, `medecin_principal`, `quantite`, `action`, `facturable`, `valide`, `modificateurs`, `extension_pmsi`, `extension_documentaire`, `association_non_prevue`.

Usage

```
write_evenements_serveur_actes(
  evenements,
  actes_ccam = NULL,
  actes_ngap = NULL,
  entete = NULL,
  out_dir,
  filename = "{evt_id}.xml",
```

```
  version = NULL,
  xmlns = HPRIM_NS,
  encoding = "UTF-8",
  validate_xsd = "error",
  colmap = NULL,
  identifiant_egal_nom_fichier = TRUE,
  fichier_ok = FALSE,
  par_acte = FALSE,
  actes = NULL
)
```

Arguments

evenements, actes_ccam, actes_ngap	Tables (actes_ccam : CCAM ; actes_ngap : NGAP).
entete	Liste : emetteur, destinataire, et en option version, mode_traitement, acquittement_attendu, commentaire, date_heure_production, emetteur_categorie, destinataire_categorie. Chaque clé peut aussi être une colonne de evenements (prioritaire).
out_dir	Dossier de sortie.
filename	Gabarit ("{evt_id}.xml") ou fonction ligne -> nom.
version	Version du schéma ; sinon entete$version ou colonne version, sinon 2.4.
xmlns	HPRIM_NS (défaut), NULL (pas de namespace) ou "" (xmlns="").
encoding	Encodage des fichiers.
validate_xsd	"error" (défaut), "warn" ou FALSE.
colmap	Renommage de colonnes list(ancien = "nouveau").
identifiant_egal_nom_fichier	L'identifiantMessage est le nom du fichier sans extension (défaut) ; une colonne identifiant_message contradictoire est une erreur.
fichier_ok	Dépose un .ok vide une fois chaque .xml écrit intégralement.
par_acte	Un message par acte (unité atomique) au lieu d'un par événement. Le contexte (patient, venue, intervention) est répété ; evt_id vaut {evt_id}_{rang}, et le gabarit filename dispose aussi de {evt_origine}, {rang} et {acte_id}.
actes	Ancien nom de actes_ccam, encore accepté avec un avertissement.

Details

Unité atomique du serveur d'actes : l'acte (acteCCAM), porteur de l'attribut action et d'un identifiant. par_acte = TRUE écrit un message par acte.

actes_ngap : une ligne par acte NGAP (consultation, forfait technique, biologie), avec evt_id, acte_id, lettre_cle, coefficient, date_execution, medecin et en option heure_execution, uf, quantite, denombrement, nabm (codes NABM : colonne liste ou chaîne séparée par des virgules), montant_total, numero_forfait_technique, numero_agrement_appareil, action, facturable, valide. Un message peut porter des actes CCAM, des actes NGAP, ou les deux ; par_acte découpe les deux.

Value

Les chemins écrits (invisible).

Examples

```
# Les exemples écrivent dans un dossier temporaire
ancien <- setwd(tempdir())
# Régénère data/xml_tests/actes/00001.xml : une intervention et quatre actes CCAM.
library(hprimxml)
library(tibble)

ts <- function(x) as.POSIXct(x, tz = "UTC")

entete <- list(
  version = "2.00",                    # version HPRIM XML des messages
  emetteur = "TEST",
  destinataire = NULL,
  acquittement_attendu = "oui",
  commentaire = "HPRIM-TEST",
  date_heure_production = ts("2024-01-01 01:01:01")
)

# Une ligne par événement : patient, venue, intervention et demande
evenements <- tibble(
  evt_id = "00001_acte",
  ipp = "123456789", ipp_recepteur = "123456789",
  sexe = "M", nom_usuel = "SANDWICK", prenom = "JOHN",
  date_naissance = as.Date("1970-01-01"),
  nda = "987654321", nda_recepteur = "987654321",
 venue_date_entree = as.Date("2024-01-01"), venue_heure_entree = hms::as_hms("08:00:00"),
  venue_uf_responsable = "4321",
  `venue/@prive` = "non",
  intervention_id = "000000000001",
 intervention_date_debut = as.Date("2024-01-01"), intervention_heure_debut = hms::as_hms("08:10:00"),
 intervention_date_fin = as.Date("2024-01-01"), intervention_heure_fin = hms::as_hms("09:00:00"),
  intervention_uf = "4321",
 demande_date = as.Date("2023-12-25"), demande_heure = hms::as_hms("02:30:00"), demande_uf = "4321"
)

# Une ligne par acte CCAM
actes_ccam <- tibble(
  evt_id = "00001_acte",
  acte_id = c("121212", "121213", "121214", "121215"),
  code = c("EBLA003", "YYYY033", "EPLF002", "NFKA006"),
  activite = 1L, phase = 0L,
  date_execution = as.Date("2024-01-01"), heure_execution = hms::as_hms("08:10:00"),
  medecin = "01234567", uf = "4321",
  quantite = "1",                      # nombre5 : chaîne de chiffres dans le XSD
  facturable = "oui", valide = "oui"
)

fichiers <- write_evenements_serveur_actes(
```

```
  evenements, actes_ccam,
  entete = entete,
  out_dir = "sortie/actes",             # -> sortie/actes/00001_acte.xml
  validate_xsd = "warn"
)
fichiers
setwd(ancien)
```

Index

∗ **datasets**
HPRIM_NS, 5

hprim_anonymiser, 2
hprim_anonymiser(), *3*, *5*, *7*
hprim_anonymiser_message, 3
hprim_available_versions, 4
hprim_decalage_jours, 4
hprim_decalage_jours(), *3*
hprim_faux_nom, 5
hprim_faux_prenom *(*hprim_faux_nom*)*, 5
HPRIM_NS, 5
hprim_parse_xml, 6
hprim_parse_xml(), *3*
hprim_pseudonyme, 7
hprim_to_xml, 7
hprim_to_xml(), *6*, *9*
hprim_validate, 9
hprim_write_xml, 9

iter_evenements_etat_patient, 10
iter_evenements_pmsi, 11
iter_evenements_serveur_actes, 12

read_evenements_etat_patient, 12
read_evenements_pmsi, 13
read_evenements_serveur_actes, 14

suivi_depots, 15

write_evenements_etat_patient, 16
write_evenements_pmsi, 21
write_evenements_pmsi(), *13*
write_evenements_serveur_actes, 25