Package 'geosapi'

October 13, 2022

Type Package

Title GeoServer REST API R Interface

Version 0.6-4

Date 2022-08-18

Maintainer Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Description Provides an R interface to the GeoServer REST API, allowing to upload and publish data in a GeoServer web-application and expose data to OGC Web-Services. The package currently supports all CRUD (Create,Read,Update,Delete) operations on GeoServer workspaces, namespaces, datastores (stores of vector data), featuretypes, layers, styles, as well as vector data upload operations. For more information about the GeoServer REST API, see <https://docs.geoserver.org/stable/en/user/rest/>.

Depends R (>= 3.1.0)

Imports R6, openssl, httr, XML, keyring, readr

Suggests testthat, roxygen2, covr, shiny, knitr, markdown

License MIT + file LICENSE

URL https://github.com/eblondel/geosapi, https://eblondel.github.io/geosapi/, https://geoserver.org/

BugReports https://github.com/eblondel/geosapi/issues

LazyLoad yes

RoxygenNote 7.2.1

VignetteBuilder knitr

NeedsCompilation no

Author Emmanuel Blondel [aut, cre] (<https://orcid.org/0000-0002-5870-5762>)

Repository CRAN

Date/Publication 2022-08-18 18:30:02 UTC

R topics documented:

`geosapi` *GeoServer REST API R Interface*

Description

Provides an R interface to the GeoServer REST API, allowing to upload and publish data in a GeoServer web-application and expose data to OGC Web-Services. The package currently supports all CRUD (Create,Read,Update,Delete) operations on GeoServer workspaces, namespaces, datastores (stores of vector data), featuretypes, layers, styles, as well as vector data upload operations. For more information about the GeoServer REST API, see <https://docs.geoserver.org/stable/en/user/rest/> .

Details

Package:	geosapi
Type:	Package
Version:	0.6
Date:	2022-02-22
License:	MIT
LazyLoad:	yes

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`GSAbstractCoverageStore`
Geoserver REST API CoverageStore

Description

Geoserver REST API CoverageStore

Geoserver REST API CoverageStore

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer CoverageStore

Super classes

geosapi::GSRESTResource -> geosapi::GSAbstractStore -> GSAbstractCoverageStore

Public fields

`url` URL of the abstract coverage store

Methods

Public methods:

- GSAbstractCoverageStore$new()
- GSAbstractCoverageStore$decode()
- GSAbstractCoverageStore$setUrl()
- GSAbstractCoverageStore$clone()

Method `new()`**:** initializes an abstract coverage store

Usage:

```
GSAbstractCoverageStore$new(
  xml = NULL,
  type = NULL,
  name = NULL,
  description = "",
  enabled = TRUE,
  url = NULL
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`type` the type of coverage store

`name` coverage store name

`description` coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

`url` URL of the store

Method `decode()`**:** Decodes a coverage store from XML

Usage:

```
GSAbstractCoverageStore$decode(xml)
```

Arguments:

`xml` an object of class XMLInternalNode-class

Returns: an object of class GSAbstractCoverageStore

Method `setUrl()`**:** set coverage store URL

Usage:

```
GSAbstractCoverageStore$setUrl(url)
```

Arguments:

`url` the store URL to set

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSAbstractCoverageStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`GSAbstractDataStore` *Geoserver REST API DataStore*

Description

Geoserver REST API DataStore

Geoserver REST API DataStore

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer dataStore

Super classes

`geosapi::GSRESTResource` -> `geosapi::GSAbstractStore` -> `GSAbstractDataStore`

Public fields

`connectionParameters` the list of connection parameters

Methods

Public methods:

- `GSAbstractDataStore$new()`
- `GSAbstractDataStore$decode()`
- `GSAbstractDataStore$setConnectionParameters()`
- `GSAbstractDataStore$addConnectionParameter()`
- `GSAbstractDataStore$setConnectionParameter()`
- `GSAbstractDataStore$delConnectionParameter()`
- `GSAbstractDataStore$clone()`

Method `new()`: initializes an abstract data store

Usage:

```
GSAbstractDataStore$new(
  xml = NULL,
  type = NULL,
  name = NULL,
  description = "",
  enabled = TRUE,
  connectionParameters
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`type` the type of coverage store

`name` coverage store name

`description` coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

`connectionParameters` the list of connection parameters

Method `decode()`: Decodes a data store from XML

Usage:

```
GSAbstractDataStore$decode(xml)
```

Arguments:

`xml` an object of class XMLInternalNode-class

Returns: an object of class GSAbstractDataStore

Method `setConnectionParameters()`: Set list connection parameters. The argument should be an object of class `GSRESTEntrySet` giving a list of key/value parameter entries.

Usage:

```
GSAbstractDataStore$setConnectionParameters(parameters)
```

Arguments:

`parameters` an object of class GSRESTEntrySet

Method `addConnectionParameter()`: Adds a connection parameter

Usage:

```
GSAbstractDataStore$addConnectionParameter(key, value)
```

Arguments:

`key` connection parameter key

`value` connection parameter value

Returns: `TRUE` if added, `FALSE` otherwise

Method `setConnectionParameter()`: Sets a connection parameter

Usage:

```
GSAbstractDataStore$setConnectionParameter(key, value)
```

Arguments:

`key` connection parameter key

`value` connection parameter value

Method `delConnectionParameter()`**:** Removes a connection parameter

Usage:

```
GSAbstractDataStore$delConnectionParameter(key)
```

Arguments:

`key` connection parameter key

`value` connection parameter value

Returns: `TRUE` if removed, `FALSE` otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSAbstractDataStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`GSAbstractDBDataStore` *Geoserver REST API AbstractDBDataStore*

Description

Geoserver REST API AbstractDBDataStore

Geoserver REST API AbstractDBDataStore

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer abstract DB dataStore

Super classes

`geosapi::GSRESTResource` -> `geosapi::GSAbstractStore` -> `geosapi::GSAbstractDataStore` -> `GSAbstractDBDataStore`

Methods

Public methods:

- GSAbstractDBDataStore$new()
- GSAbstractDBDataStore$setDatabaseType()
- GSAbstractDBDataStore$setNamespace()
- GSAbstractDBDataStore$setHost()
- GSAbstractDBDataStore$setPort()
- GSAbstractDBDataStore$setDatabase()
- GSAbstractDBDataStore$setSchema()
- GSAbstractDBDataStore$setUser()
- GSAbstractDBDataStore$setPassword()
- GSAbstractDBDataStore$setJndiReferenceName()
- GSAbstractDBDataStore$setExposePrimaryKeys()
- GSAbstractDBDataStore$setMinConnections()
- GSAbstractDBDataStore$setMaxConnections()
- GSAbstractDBDataStore$setFetchSize()
- GSAbstractDBDataStore$setConnectionTimeout()
- GSAbstractDBDataStore$setValidateConnections()
- GSAbstractDBDataStore$setPrimaryKeyMetadataTable()
- GSAbstractDBDataStore$setLooseBBox()
- GSAbstractDBDataStore$setPreparedStatements()
- GSAbstractDBDataStore$setMaxOpenPreparedStatements()
- GSAbstractDBDataStore$setEstimatedExtends()
- GSAbstractDBDataStore$setDefautConnectionParameters()
- GSAbstractDBDataStore$clone()

Method `new()`**:** initializes an abstract DB data store

Usage:

```
GSAbstractDBDataStore$new(
  xml = NULL,
  type = NULL,
  dbType = NULL,
  name = NULL,
  description = "",
  enabled = TRUE
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`type` the type of DB data store

`dbType` DB type

`name` coverage store name

`description` coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

Method `setDatabaseType()`: Set database type

Usage:

```
GSAbstractDBDataStore$setDatabaseType(dbtype)
```

Arguments:

`dbtype` DB type

Method `setNamespace()`: Set namespace

Usage:

```
GSAbstractDBDataStore$setNamespace(namespace)
```

Arguments:

`namespace` namespace

Method `setHost()`: Set host

Usage:

```
GSAbstractDBDataStore$setHost(host)
```

Arguments:

`host` host

Method `setPort()`: Set port

Usage:

```
GSAbstractDBDataStore$setPort(port)
```

Arguments:

`port` port

Method `setDatabase()`: Set database

Usage:

```
GSAbstractDBDataStore$setDatabase(database)
```

Arguments:

`database` database

Method `setSchema()`: Set schema

Usage:

```
GSAbstractDBDataStore$setSchema(schema)
```

Arguments:

`schema` schema

Method `setUser()`: Set user

Usage:

```
GSAbstractDBDataStore$setUser(user)
```

Arguments:

`user` user

Method `setPassword()`: Set password

Usage:

```
GSAbstractDBDataStore$setPassword(password)
```

Arguments:

`password` password

Method `setJndiReferenceName()`: Set JNDI reference name

Usage:

```
GSAbstractDBDataStore$setJndiReferenceName(jndiReferenceName)
```

Arguments:

`jndiReferenceName` JNDI reference name

Method `setExposePrimaryKeys()`: Set expose primary keyws

Usage:

```
GSAbstractDBDataStore$setExposePrimaryKeys(exposePrimaryKeys)
```

Arguments:

`exposePrimaryKeys` expose primary keys

Method `setMinConnections()`: Set min connections

Usage:

```
GSAbstractDBDataStore$setMinConnections(minConnections = 1)
```

Arguments:

`minConnections` min connections. Default is 11

Method `setMaxConnections()`: Set max connections

Usage:

```
GSAbstractDBDataStore$setMaxConnections(maxConnections = 10)
```

Arguments:

`maxConnections` max connections. Default is 10

Method `setFetchSize()`: Set fetch size

Usage:

```
GSAbstractDBDataStore$setFetchSize(fetchSize = 1000)
```

Arguments:

`fetchSize` fetch size. Default is 1000

Method `setConnectionTimeout()`: Set connection timeout

Usage:

```
GSAbstractDBDataStore$setConnectionTimeout(seconds = 20)
```

Arguments:

`seconds` timeout (in seconds). Default is 20

Method `setValidateConnections()`: Set validate connection

Usage:

```
GSAbstractDBDataStore$setValidateConnections(validateConnections)
```

Arguments:

`validateConnections` Validate connections

Method `setPrimaryKeyMetadataTable()`: Set primary key metadata table

Usage:

```
GSAbstractDBDataStore$setPrimaryKeyMetadataTable(primaryKeyMetadataTable)
```

Arguments:

`primaryKeyMetadataTable` primary key metadata table

Method `setLooseBBox()`: Set loose bbox

Usage:

```
GSAbstractDBDataStore$setLooseBBox(looseBBox = TRUE)
```

Arguments:

`looseBBox` loose bbox. Default is `TRUE`

Method `setPreparedStatements()`: Set prepared statemnts

Usage:

```
GSAbstractDBDataStore$setPreparedStatements(preparedStatements = FALSE)
```

Arguments:

`preparedStatements` prepared Statements. Default is `FALSE`

Method `setMaxOpenPreparedStatements()`: Set max open prepared statements

Usage:

```
GSAbstractDBDataStore$setMaxOpenPreparedStatements(
  maxOpenPreparedStatements = 50
)
```

Arguments:

`maxOpenPreparedStatements` max open preepared statements. Default is 50

Method `setEstimatedExtends()`: Set estimatedExtends

Usage:

```
GSAbstractDBDataStore$setEstimatedExtends(estimatedExtends = FALSE)
```

Arguments:

`estimatedExtends` estimated extends. Default is `FALSE`

Method `setDefautConnectionParameters()`: Set default connection parameters

Usage:

```
GSAbstractDBDataStore$setDefautConnectionParameters()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
GSAbstractDBDataStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

Internal abstract class used for setting DB stores

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`GSAbstractStore` *Geoserver REST API Store*

Description

Geoserver REST API Store

Geoserver REST API Store

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer store

Super class

`geosapi::GSRESTResource` -> `GSAbstractStore`

Public fields

`full` whether store object is fully described

`name` store name

`enabled` if the store is enabled or not

`description` store description

`type` store type

`workspace` workspace name

Methods

Public methods:

- `GSAbstractStore$new()`
- `GSAbstractStore$decode()`
- `GSAbstractStore$setType()`
- `GSAbstractStore$setEnabled()`
- `GSAbstractStore$setDescription()`
- `GSAbstractStore$clone()`

Method `new()`: initializes an abstract store

Usage:

```
GSAbstractStore$new(
  xml = NULL,
  storeType,
  type = NULL,
  name = NULL,
  description = "",
  enabled = TRUE
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`storeType` store type

`type` the type of coverage store

`name` coverage store name

`description` coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

Method `decode()`: Decodes store from XML

Usage:

```
GSAbstractStore$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setType()`: Set type

Usage:

```
GSAbstractStore$setType(type)
```

Arguments:

`type` type

Method `setEnabled()`: Set enabled

Usage:

```
GSAbstractStore$setEnabled(enabled)
```

Arguments:

`enabled` enabled

Method `setDescription()`: Set description

Usage:

```
GSAbstractStore$setDescription(description)
```

Arguments:

`description` description

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
GSAbstractStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`GSArcGridCoverageStore`

Geoserver REST API ArcGridCoverageStore

Description

Geoserver REST API ArcGridCoverageStore

Geoserver REST API ArcGridCoverageStore

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer ArcGrid CoverageStore

Super classes

`geosapi::GSRESTResource` -> `geosapi::GSAbstractStore` -> `geosapi::GSAbstractCoverageStore` -> `GSArcGridCoverageStore`

Public fields

`url` url

Methods

Public methods:

- `GSArcGridCoverageStore$new()`
- `GSArcGridCoverageStore$clone()`

Method `new()`**:** initializes an abstract ArcGrid coverage store

Usage:

```
GSArcGridCoverageStore$new(
  xml = NULL,
  name = NULL,
  description = "",
  enabled = TRUE,
  url = NULL
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`name` coverage store name

`description` coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

`url` url

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSArcGridCoverageStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`GSCoverage` *A GeoServer coverage*

Description

This class models a GeoServer coverage. This class is to be used for manipulating representations of vector data with GeoServer.

Format

`R6Class` object.

Details

Geoserver REST API Resource

Value

Object of `R6Class` for modelling a GeoServer coverage

Super classes

`geosapi::GSRESTResource` -> `geosapi::GSResource` -> `GSCoverage`

Public fields

`cqlFilter` CQL filter

Methods

Public methods:

- GSCoverage$new()
- GSCoverage$decode()
- GSCoverage$setView()
- GSCoverage$delView()
- GSCoverage$clone()

Method `new()`**:** Initializes a GSCoverage from XML

Usage:

```
GSCoverage$new(xml = NULL)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `decode()`**:** Decodes coverage from XML

Usage:

```
GSCoverage$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setView()`**:** Set view

Usage:

```
GSCoverage$setView(cv)
```

Arguments:

`cv` cv, object of class GSCoverageView

Returns: `TRUE` if set, `FALSE` otherwise

Method `delView()`**:** Deletes view

Usage:

```
GSCoverage$delView()
```

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSCoverage$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
gt <- GSCoverage$new()
```

GSCoverageBand *Geoserver REST API GSCoverageBand*

Description

Geoserver REST API GSCoverageBand

Geoserver REST API GSCoverageBand

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer coverage band

Methods

`new(xml)` This method is used to instantiate a `GSCoverageBand`

`decode(xml)` This method is used to decode a `GSCoverageBand` from XML

`encode()` This method is used to encode a `GSCoverageBand` to XML

`setDefinition(definition)` Sets the coverage band definition

`setIndex(index)` Sets the coverage band index

`setCompositionType` Sets the composition type. Only 'BAND_SELECT' is supported by GeoServer for now.

`addInputBand(band)` Adds a input coverage band, object of class `GSInputCoverageBand`

`delInputBand(band)` Removes a input coverage band, object of class `GSInputCoverageBand`

Super class

`geosapi::GSRESTResource` -> `GSCoverageBand`

Public fields

`inputCoverageBands` list of input coverage bands

`definition` coverage band definition

`index` coverage band index

`compositionType` coverage band composition type

Methods

Public methods:

- GSCoverageBand$new()
- GSCoverageBand$decode()
- GSCoverageBand$setName()
- GSCoverageBand$setDefinition()
- GSCoverageBand$setIndex()
- GSCoverageBand$setCompositionType()
- GSCoverageBand$addInputBand()
- GSCoverageBand$delInputBand()
- GSCoverageBand$clone()

Method `new()`**:** Initalizes a GSCoverageBand

Usage:

```
GSCoverageBand$new(xml = NULL)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `decode()`**:** Decodes from XML

Usage:

```
GSCoverageBand$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setName()`**:** Set name

Usage:

```
GSCoverageBand$setName(name)
```

Arguments:

`name` name

Method `setDefinition()`**:** Set definition

Usage:

```
GSCoverageBand$setDefinition(definition)
```

Arguments:

`definition` definition

Method `setIndex()`**:** Set index

Usage:

```
GSCoverageBand$setIndex(index)
```

Arguments:

`index` index

Method `setCompositionType()`**:** Set composition type

Usage:

```
GSCoverageBand$setCompositionType(compositionType)
```

Arguments:

`compositionType` composition type

Method `addInputBand()`**:** Adds an input band

Usage:

```
GSCoverageBand$addInputBand(band)
```

Arguments:

`band` object of class GSInputCoverageBand

Returns: `TRUE` if added, `FALSE` otherwise

Method `delInputBand()`**:** Deletes an input band

Usage:

```
GSCoverageBand$delInputBand(band)
```

Arguments:

`band` object of class GSInputCoverageBand

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSCoverageBand$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSCoverageBand$new()
```

GSCoverageStoreManager

Geoserver REST API CoverageStore Manager

Description

Geoserver REST API CoverageStore Manager

Geoserver REST API CoverageStore Manager

Format

R6Class object.

Value

Object of R6Class with methods for managing GeoServer CoverageStores (i.e. stores of coverage data)

Super class

geosapi::GSManager -> GSCoverageStoreManager

Methods

Public methods:

- GSCoverageStoreManager$getCoverageStores()
- GSCoverageStoreManager$getCoverageStoreNames()
- GSCoverageStoreManager$getCoverageStore()
- GSCoverageStoreManager$createCoverageStore()
- GSCoverageStoreManager$updateCoverageStore()
- GSCoverageStoreManager$deleteCoverageStore()
- GSCoverageStoreManager$getCoverages()
- GSCoverageStoreManager$getCoverageNames()
- GSCoverageStoreManager$getCoverage()
- GSCoverageStoreManager$createCoverage()
- GSCoverageStoreManager$updateCoverage()
- GSCoverageStoreManager$deleteCoverage()
- GSCoverageStoreManager$uploadCoverage()
- GSCoverageStoreManager$uploadGeoTIFF()
- GSCoverageStoreManager$uploadWorldImage()
- GSCoverageStoreManager$uploadArcGrid()
- GSCoverageStoreManager$uploadImageMosaic()
- GSCoverageStoreManager$clone()

Method `getCoverageStores()`**:** Get the list of available coverage stores. Returns an object of class `list` giving items of class `GSAbstractCoverageStore`

Usage:

```
GSCoverageStoreManager$getCoverageStores(ws)
```

Arguments:

`ws` workspace name

Returns: the list of coverage stores

Method `getCoverageStoreNames()`**:** Get the list of available coverage store names. Returns an vector of class `character`

Usage:

```
GSCoverageStoreManager$getCoverageStoreNames(ws)
```

Arguments:

`ws` workspace name

Returns: the list of coverage store names, as `character`

Method `getCoverageStore()`**:** Get an object of class `GSAbstractDataStore` given a workspace and coverage store names.

Usage:

```
GSCoverageStoreManager$getCoverageStore(ws, cs)
```

Arguments:

`ws` workspace name

`cs` coverage store name

Returns: the coverage store

Method `createCoverageStore()`**:** Creates a new coverage store given a workspace, coverage store name. Abstract method used in below format-specific methods to create coverage stores.

Usage:

```
GSCoverageStoreManager$createCoverageStore(ws, coverageStore)
```

Arguments:

`ws` workspace name

`coverageStore` coverage store object

Returns: `TRUE` if created, `FALSE` otherwise

Method `updateCoverageStore()`**:** Updates a coverage store given a workspace, coverage store name. Abstract method used in below format-specific methods to create coverage stores.

Usage:

```
GSCoverageStoreManager$updateCoverageStore(ws, coverageStore)
```

Arguments:

`ws` workspace name

`coverageStore` coverage store object

Returns: `TRUE` if updated, `FALSE` otherwise

Method `deleteCoverageStore()`**:** Deletes a coverage store given a workspace and an object of class `GSAbstractCoverageStore`. By defaut, the option `recurse` is set to FALSE, ie datastore layers are not removed. To remove all coverage store layers, set this option to TRUE. The `purge` parameter is used to customize the delete of files on disk (in case the underlying reader implements a delete method). It can take one of the three values: none, metadata, all. For more details see https://docs.geoserver.org/stable/en/user/rest/api/coveragestores.html#purge

Usage:

```
GSCoverageStoreManager$deleteCoverageStore(
  ws,
  cs,
  recurse = FALSE,
  purge = NULL
)
```

Arguments:

`ws` workspace name

`cs` coverage store name

`recurse` recurse

`purge` purge

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `getCoverages()`**:** Get the list of available coverages for given workspace and coverage store. Returns an object of class `list` giving items of class `GSCoverage`

Usage:

```
GSCoverageStoreManager$getCoverages(ws, cs)
```

Arguments:

`ws` workspace name

`cs` coverage store name

Returns: the list of GSCoverage

Method `getCoverageNames()`**:** Get the list of available coverage names for given workspace and coverage store. Returns an object of class `list` giving items of class `GSCoverage`

Usage:

```
GSCoverageStoreManager$getCoverageNames(ws, cs)
```

Arguments:

`ws` workspace name

`cs` coverage store name

Returns: the list of coverage names

Method `getCoverage()`**:** Get coverage

Usage:

```
GSCoverageStoreManager$getCoverage(ws, cs, cv)
```

Arguments:

`ws` workspace name

cs coverage store name

cv coverage name

Method createCoverage(): Creates a new coverage given a workspace, coverage store names and an object of class GSCoverage

Usage:

```
GSCoverageStoreManager$createCoverage(ws, cs, coverage)
```

Arguments:

ws workspace name

cs coverage store name

coverage object of class GSCoverage

Returns: TRUE if created, FALSE otherwise

Method updateCoverage(): Updates a coverage given a workspace, coverage store names and an object of class GSCoverage

Usage:

```
GSCoverageStoreManager$updateCoverage(ws, cs, coverage)
```

Arguments:

ws workspace name

cs coverage store name

coverage object of class GSCoverage

Returns: TRUE if updated, FALSE otherwise

Method deleteCoverage(): Deletes a coverage given a workspace, coverage store names, and an object of class GSCoverage. By defaut, the option recurse is set to FALSE, ie coverage layers are not removed.

Usage:

```
GSCoverageStoreManager$deleteCoverage(ws, cs, cv, recurse = FALSE)
```

Arguments:

ws workspace name

cs coverage store name

cv coverage name

recurse recurse

Method uploadCoverage(): Abstract method to upload a coverage file targeting a workspace (ws) and datastore (cs). The extension corresponds to the format/type of coverage to be uploaded (among values 'geotiff', 'worldimage', 'arcgrid', or 'imagemosaic'). The endpoint takes a value among "file" (default), "url" or "external". The filename is the name of the coverage file to upload and set for the newly created datastore. The configure parameter can take a value among values "none" (indicates to configure only the datastore but no layer configuration) or "first" (configure both datastore and layer). The update defines the strategy for the upload: "append" (default value) for the first upload, "overwrite" in case the file should be overwriten.

Usage:

```
GSCoverageStoreManager$uploadCoverage(
  ws,
  cs,
  endpoint = "file",
  extension,
  filename,
  configure = "first",
  update = "append",
  contentType
)
```

Arguments:

`ws` workspace name

`cs` coverage store name

`endpoint` endpoint. Default is "file"

`extension` extension

`filename` filename

`configure` configure. Default is "first"

`update` update. Default is "append"

`contentType` content type

Returns: `TRUE` if uploaded, `FALSE` otherwise

Method `uploadGeoTIFF()`**:** Uploads a GeoTIFF file targeting a workspace (`ws`) and datastore (`cs`). The `endpoint` takes a value among `"file"` (default), `"url"` or `"external"`. The `filename` is the name of the GeoTIFF file to upload and set for the newly created datastore. The `configure` parameter can take a value among values `"none"` (indicates to configure only the datastore but no layer configuration) or `"first"` (configure both datastore and layer). The `update` defines the strategy for the upload: `"append"` (default value) for the first upload, `"overwrite"` in case the file should be overwriten.

Usage:

```
GSCoverageStoreManager$uploadGeoTIFF(
  ws,
  cs,
  endpoint = "file",
  filename,
  configure = "first",
  update = "append"
)
```

Arguments:

`ws` workspace name

`cs` coverage store name

`endpoint` endpoint. Default is "file"

`filename` filename

`configure` configure. Default is "first"

`update` update. Default is "append"

Returns: `TRUE` if uploaded, `FALSE` otherwise

Method `uploadWorldImage()`**:** Uploads a WorldImage file targeting a workspace (`ws`) and datastore (`cs`). The `endpoint` takes a value among `"file"` (default), `"url"` or `"external"`. The `filename` is the name of the zipped file to upload and set for the newly created datastore. It is assumed the zip archive contains the .prj file to set the SRS. The `configure` parameter can take a value among values `"none"` (indicates to configure only the datastore but no layer configuration) or `"first"` (configure both datastore and layer). The `update` defines the strategy for the upload: `"append"` (default value) for the first upload, `"overwrite"` in case the file should be overwriten.

Usage:

```
GSCoverageStoreManager$uploadWorldImage(
  ws,
  cs,
  endpoint = "file",
  filename,
  configure = "first",
  update = "append"
)
```

Arguments:

`ws` workspace name

`cs` coverage store name

`endpoint` endpoint. Default is "file"

`filename` filename

`configure` configure. Default is "first"

`update` update. Default is "append"

Returns: `TRUE` if uploaded, `FALSE` otherwise

Method `uploadArcGrid()`**:** Uploads an ArcGrid file targeting a workspace (`ws`) and datastore (`cs`). The `endpoint` takes a value among `"file"` (default), `"url"` or `"external"`. The `filename` is the name of the ArcGrid file to upload and set for the newly created datastore. The `configure` parameter can take a value among values `"none"` (indicates to configure only the datastore but no layer configuration) or `"first"` (configure both datastore and layer). The `update` defines the strategy for the upload: `"append"` (default value) for the first upload, `"overwrite"` in case the file should be overwriten.

Usage:

```
GSCoverageStoreManager$uploadArcGrid(
  ws,
  cs,
  endpoint = "file",
  filename,
  configure = "first",
  update = "append"
)
```

Arguments:

`ws` workspace name

`cs` coverage store name

`endpoint` endpoint. Default is "file"

filename filename

configure configure. Default is "first"

update update. Default is "append"

Returns: TRUE if uploaded, FALSE otherwise

Method uploadImageMosaic(): Uploads an ImageMosaic file targeting a workspace (ws) and datastore (cs). The endpoint takes a value among "file" (default), "url" or "external". The filename is the name of the ImageMosaic file to upload and set for the newly created datastore. The configure parameter can take a value among values "none" (indicates to configure only the datastore but no layer configuration) or "first" (configure both datastore and layer). The update defines the strategy for the upload: "append" (default value) for the first upload, "overwrite" in case the file should be overwriten.

Usage:

```
GSCoverageStoreManager$uploadImageMosaic(
  ws,
  cs,
  endpoint = "file",
  filename,
  configure = "first",
  update = "append"
)
```

Arguments:

ws workspace name

cs coverage store name

endpoint endpoint. Default is "file"

filename filename

configure configure. Default is "first"

update update. Default is "append"

Returns: TRUE if uploaded, FALSE otherwise

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
GSCoverageStoreManager$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
## Not run:
   GSCoverageStoreManager$new("http://localhost:8080/geoserver", "admin", "geoserver")

## End(Not run)
```

GSCoverageView *Geoserver REST API GSCoverageView*

Description

Geoserver REST API GSCoverageView

Geoserver REST API GSCoverageView

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer coverage view

Super class

geosapi::GSRESTResource -> GSCoverageView

Public fields

`name` name

`envelopeCompositionType` envelope composition type

`selectedResolution` selected resolution

`selectedResolutionIndex` selected resolution index

`coverageBands` coverage bands

Methods

Public methods:

- GSCoverageView$new()
- GSCoverageView$decode()
- GSCoverageView$setName()
- GSCoverageView$setEnvelopeCompositionType()
- GSCoverageView$setSelectedResolution()
- GSCoverageView$setSelectedResolutionIndex()
- GSCoverageView$addBand()
- GSCoverageView$delBand()
- GSCoverageView$clone()

Method `new()`**:** Initializes an object of class GSCoverageView

Usage:

```
GSCoverageView$new(xml = NULL)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `decode()`: Decodes from XML

Usage:

```
GSCoverageView$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setName()`: Set name

Usage:

```
GSCoverageView$setName(name)
```

Arguments:

`name` name

Method `setEnvelopeCompositionType()`: Sets the envelope composition type. Type of Envelope Composition, used to expose the bounding box of the CoverageView, either 'UNION' or 'INTERSECTION'.

Usage:

```
GSCoverageView$setEnvelopeCompositionType(envelopeCompositionType)
```

Arguments:

`envelopeCompositionType` envelope composition type

Method `setSelectedResolution()`: Set selected resolution

Usage:

```
GSCoverageView$setSelectedResolution(selectedResolution)
```

Arguments:

`selectedResolution` selected resolution

Method `setSelectedResolutionIndex()`: Set selected resolution index

Usage:

```
GSCoverageView$setSelectedResolutionIndex(selectedResolutionIndex)
```

Arguments:

`selectedResolutionIndex` selected resolution index

Method `addBand()`: Adds band

Usage:

```
GSCoverageView$addBand(band)
```

Arguments:

`band` object of class GSCoverageBand

Returns: `TRUE` if added, `FALSE` otherwise

Method `delBand()`: Deletes band

Usage:

```
GSCoverageView$delBand(band)
```

Arguments:

band object of class GSCoverageBand

Returns: TRUE if deleted, FALSE otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSCoverageView$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSCoverageView$new()
```

`GSDataStoreManager` *Geoserver REST API DataStore Manager*

Description

Geoserver REST API DataStore Manager

Geoserver REST API DataStore Manager

Format

`R6Class` object.

Value

Object of `R6Class` with methods for managing GeoServer DataStores (i.e. stores of vector data)

Super class

`geosapi::GSManager` -> `GSDataStoreManager`

Methods

Public methods:

- GSDataStoreManager$getDataStores()
- GSDataStoreManager$getDataStoreNames()
- GSDataStoreManager$getDataStore()
- GSDataStoreManager$createDataStore()
- GSDataStoreManager$updateDataStore()
- GSDataStoreManager$deleteDataStore()
- GSDataStoreManager$getFeatureTypes()
- GSDataStoreManager$getFeatureTypeNames()
- GSDataStoreManager$getFeatureType()
- GSDataStoreManager$createFeatureType()
- GSDataStoreManager$updateFeatureType()
- GSDataStoreManager$deleteFeatureType()
- GSDataStoreManager$publishLayer()
- GSDataStoreManager$unpublishLayer()
- GSDataStoreManager$uploadData()
- GSDataStoreManager$uploadShapefile()
- GSDataStoreManager$uploadProperties()
- GSDataStoreManager$uploadH2()
- GSDataStoreManager$uploadSpatialite()
- GSDataStoreManager$uploadAppschema()
- GSDataStoreManager$uploadGeoPackage()
- GSDataStoreManager$clone()

Method `getDataStores()`**:** Get the list of available dataStores.

Usage:

```
GSDataStoreManager$getDataStores(ws)
```

Arguments:

`ws` workspace name

Returns: an object of class `list` giving items of class `GSAbstractDataStore`

Method `getDataStoreNames()`**:** Get the list of available dataStore names.

Usage:

```
GSDataStoreManager$getDataStoreNames(ws)
```

Arguments:

`ws` workspace name

Returns: a vector of class `character`

Method `getDataStore()`**:** Get an object of class `GSAbstractDataStore` given a workspace and datastore names.

Usage:

```
GSDataStoreManager$getDataStore(ws, ds)
```

Arguments:

`ws` workspace name

`ds` datastore name

Returns: the datastore

Method `createDataStore()`**:** Creates a datastore given a workspace and an object of class `GSAbstractDataStore`.

Usage:

```
GSDataStoreManager$createDataStore(ws, dataStore)
```

Arguments:

`ws` workspace name

`dataStore` datastore object of class GSAbstractDataStore

Returns: `TRUE` if created, `FALSE` otherwise

Method `updateDataStore()`**:** Updates a datastore given a workspace and an object of class `GSAbstractDataStore`.

Usage:

```
GSDataStoreManager$updateDataStore(ws, dataStore)
```

Arguments:

`ws` workspace name

`dataStore` datastore object of class GSAbstractDataStore

Returns: `TRUE` if updated, `FALSE` otherwise

Method `deleteDataStore()`**:** Deletes a datastore given workspace and datastore names. By defaut, the option `recurse` is set to FALSE, ie datastore layers are not removed. To remove all datastore layers, set this option to TRUE.

Usage:

```
GSDataStoreManager$deleteDataStore(ws, ds, recurse = FALSE)
```

Arguments:

`ws` workspace name

`ds` datastore name

`recurse` recurse

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `getFeatureTypes()`**:** Get the list of available feature types for given workspace and datastore.

Usage:

```
GSDataStoreManager$getFeatureTypes(ws, ds, list = "configured")
```

Arguments:

`ws` workspace name

`ds` datastore name

`list` list type value, among "configured", "available", "available_with_geom", "all"

Returns: an object of class `list` giving items of class `GSFeatureType`

Method `getFeatureTypeNames()`**:** Get the list of available feature type names for given workspace and datastore.

Usage:

```
GSDataStoreManager$getFeatureTypeNames(ws, ds)
```

Arguments:

`ws` workspace name

`ds` datastore name

Returns: a vector of class`character`

Method `getFeatureType()`**:** Get an object of class `GSFeatureType` given a workspace, datastore and feature type names.

Usage:

```
GSDataStoreManager$getFeatureType(ws, ds, ft)
```

Arguments:

`ws` workspace name

`ds` datastore name

`ft` feature type name

Returns: an object of class GSFeatureType

Method `createFeatureType()`**:** Creates a new featureType given a workspace, datastore names and an object of class `GSFeatureType`

Usage:

```
GSDataStoreManager$createFeatureType(ws, ds, featureType)
```

Arguments:

`ws` workspace name

`ds` datastore name

`featureType` feature type

Returns: `TRUE` if created, `FALSE` otherwise

Method `updateFeatureType()`**:** Updates a featureType given a workspace, datastore names and an object of class `GSFeatureType`

Usage:

```
GSDataStoreManager$updateFeatureType(ws, ds, featureType)
```

Arguments:

`ws` workspace name

`ds` datastore name

`featureType` feature type

Returns: `TRUE` if updated, `FALSE` otherwise

Method `deleteFeatureType()`**:** Deletes a featureType given a workspace, datastore names, and an object of class `GSFeatureType`. By defaut, the option `recurse` is set to FALSE, ie datastore layers are not removed.

Usage:

```
GSDataStoreManager$deleteFeatureType(ws, ds, ft, recurse = FALSE)
```

Arguments:

`ws` workspace name

`ds` datastore name

`ft` feature type name

`recurse` recurse

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `publishLayer()`**:** Publish a feature type/layer pair given a workspace and datastore. The name 'layer' here encompasses both GSFeatureType and GSLayer resources.

Usage:

```
GSDataStoreManager$publishLayer(ws, ds, featureType, layer)
```

Arguments:

`ws` workspace name

`ds` datastore name

`featureType` object of class GSFeatureType

`layer` object of class GSLayer

Returns: `TRUE` if published, `FALSE` otherwise

Method `unpublishLayer()`**:** Unpublish a feature type/layer pair given a workspace and datastore. The name 'layer' here encompasses both GSFeatureType and GSLayer resources.

Usage:

```
GSDataStoreManager$unpublishLayer(ws, ds, lyr)
```

Arguments:

`ws` workspace name

`ds` datastore name

`lyr` layer name

Returns: `TRUE` if published, `FALSE` otherwise

Method `uploadData()`**:** Uploads features data. The `extension` corresponds to the format/type of features to be uploaded among "shp", "spatialite", "h2", "gpkg". The `endpoint` takes a value among `"file"` (default), `"url"` or `"external"`. The `filename` is the name of the coverage file to upload and set for the newly created datastore. The `configure` parameter can take a value among values `"none"` (indicates to configure only the datastore but no layer configuration) or `"first"` (configure both datastore and layer). The `update` defines the strategy for the upload: `"append"` (default value) for the first upload, `"overwrite"` in case the file should be overwriten.

Usage:

```
GSDataStoreManager$uploadData(
  ws,
  ds,
  endpoint = "file",
  extension,
  configure = "first",
  update = "append",
  filename,
  charset,
  contentType
)
```

Arguments:

`ws` workspace name

`ds` datastore name

`endpoint` endpoint

`extension` extension

`configure` configure strategy among values: "first" or "none"

`update` update strategy, among values: "append", "overwrite"

`filename` file name of the resource to upload

`charset` charset

`contentType` content type

Returns: `TRUE` if uploaded, `FALSE` otherwise

Method `uploadShapefile()`**:** Uploads zipped shapefile. The `endpoint` takes a value among `"file"` (default), `"url"` or `"external"`. The `filename` is the name of the coverage file to upload and set for the newly created datastore. The `configure` parameter can take a value among values `"none"` (indicates to configure only the datastore but no layer configuration) or `"first"` (configure both datastore and layer). The `update` defines the strategy for the upload: `"append"` (default value) for the first upload, `"overwrite"` in case the file should be overwriten.

Usage:

```
GSDataStoreManager$uploadShapefile(
  ws,
  ds,
  endpoint = "file",
  configure = "first",
  update = "append",
  filename,
  charset = "UTF-8"
)
```

Arguments:

`ws` workspace name

`ds` datastore name

`endpoint` endpoint

`configure` configure strategy among values: "first" or "none"

`update` update strategy, among values: "append", "overwrite"

filename file name of the resource to upload

charset charset

Returns: TRUE if uploaded, FALSE otherwise

Method uploadProperties(): Uploads properties. The endpoint takes a value among "file" (default), "url" or "external". The filename is the name of the coverage file to upload and set for the newly created datastore. The configure parameter can take a value among values "none" (indicates to configure only the datastore but no layer configuration) or "first" (configure both datastore and layer). The update defines the strategy for the upload: "append" (default value) for the first upload, "overwrite" in case the file should be overwriten.

Usage:

```
GSDataStoreManager$uploadProperties(
  ws,
  ds,
  endpoint = "file",
  configure = "first",
  update = "append",
  filename,
  charset = "UTF-8"
)
```

Arguments:

ws workspace name

ds datastore name

endpoint endpoint

configure configure strategy among values: "first" or "none"

update update strategy, among values: "append", "overwrite"

filename file name of the resource to upload

charset charset

Returns: TRUE if uploaded, FALSE otherwise

Method uploadH2(): Uploads H2 database. The endpoint takes a value among "file" (default), "url" or "external". The filename is the name of the coverage file to upload and set for the newly created datastore. The configure parameter can take a value among values "none" (indicates to configure only the datastore but no layer configuration) or "first" (configure both datastore and layer). The update defines the strategy for the upload: "append" (default value) for the first upload, "overwrite" in case the file should be overwriten.

Usage:

```
GSDataStoreManager$uploadH2(
  ws,
  ds,
  endpoint = "file",
  configure = "first",
  update = "append",
  filename,
  charset = "UTF-8"
)
```

Arguments:

`ws` workspace name

`ds` datastore name

`endpoint` endpoint

`configure` configure strategy among values: "first" or "none"

`update` update strategy, among values: "append", "overwrite"

`filename` file name of the resource to upload

`charset` charset

Returns: `TRUE` if uploaded, `FALSE` otherwise

Method `uploadSpatialite()`**:** Uploads spatialite file. The `endpoint` takes a value among `"file"` (default), `"url"` or `"external"`. The `filename` is the name of the coverage file to upload and set for the newly created datastore. The `configure` parameter can take a value among values `"none"` (indicates to configure only the datastore but no layer configuration) or `"first"` (configure both datastore and layer). The `update` defines the strategy for the upload: `"append"` (default value) for the first upload, `"overwrite"` in case the file should be overwriten.

Usage:

```
GSDataStoreManager$uploadSpatialite(
  ws,
  ds,
  endpoint = "file",
  configure = "first",
  update = "append",
  filename,
  charset = "UTF-8"
)
```

Arguments:

`ws` workspace name

`ds` datastore name

`endpoint` endpoint

`configure` configure strategy among values: "first" or "none"

`update` update strategy, among values: "append", "overwrite"

`filename` file name of the resource to upload

`charset` charset

Returns: `TRUE` if uploaded, `FALSE` otherwise

Method `uploadAppschema()`**:** Uploads App schema. The `endpoint` takes a value among `"file"` (default), `"url"` or `"external"`. The `filename` is the name of the coverage file to upload and set for the newly created datastore. The `configure` parameter can take a value among values `"none"` (indicates to configure only the datastore but no layer configuration) or `"first"` (configure both datastore and layer). The `update` defines the strategy for the upload: `"append"` (default value) for the first upload, `"overwrite"` in case the file should be overwriten.

Usage:

```
GSDataStoreManager$uploadAppschema(
  ws,
  ds,
  endpoint = "file",
  configure = "first",
  update = "append",
  filename,
  charset = "UTF-8"
)
```

Arguments:

`ws` workspace name

`ds` datastore name

`endpoint` endpoint

`configure` configure strategy among values: "first" or "none"

`update` update strategy, among values: "append", "overwrite"

`filename` file name of the resource to upload

`charset` charset

Returns: `TRUE` if uploaded, `FALSE` otherwise

Method `uploadGeoPackage()`**:** Uploads GeoPackage. The `endpoint` takes a value among `"file"` (default), `"url"` or `"external"`. The `filename` is the name of the coverage file to upload and set for the newly created datastore. The `configure` parameter can take a value among values `"none"` (indicates to configure only the datastore but no layer configuration) or `"first"` (configure both datastore and layer). The `update` defines the strategy for the upload: `"append"` (default value) for the first upload, `"overwrite"` in case the file should be overwriten.

Usage:

```
GSDataStoreManager$uploadGeoPackage(
  ws,
  ds,
  endpoint = "file",
  configure = "first",
  update = "append",
  filename,
  charset = "UTF-8"
)
```

Arguments:

`ws` workspace name

`ds` datastore name

`endpoint` endpoint

`configure` configure strategy among values: "first" or "none"

`update` update strategy, among values: "append", "overwrite"

`filename` file name of the resource to upload

`charset` charset

Returns: `TRUE` if uploaded, `FALSE` otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSDataStoreManager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
## Not run:
   GSDataStoreManager$new("http://localhost:8080/geoserver", "admin", "geoserver")

## End(Not run)
```

`GSDimension` *A GeoServer dimension*

Description

This class models a GeoServer resource dimension.

Format

`R6Class` object.

Details

Geoserver REST API Dimension

Value

Object of `R6Class` for modelling a GeoServer dimension

Super class

`geosapi::GSRESTResource` -> `GSDimension`

Public fields

`enabled` true/false

`presentation` dimension presentation

`resolution` dimension resolution

`units` dimension units

`unitSymbol` dimension unitsSymbol

Methods

Public methods:

- GSDimension$new()
- GSDimension$decode()
- GSDimension$setEnabled()
- GSDimension$setPresentation()
- GSDimension$setUnit()
- GSDimension$setUnitSymbol()
- GSDimension$clone()

Method `new()`**:** Initializes an object of class GSDimension

Usage:

```
GSDimension$new(xml = NULL)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `decode()`**:** Decodes from XML

Usage:

```
GSDimension$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setEnabled()`**:** Set enabled

Usage:

```
GSDimension$setEnabled(enabled)
```

Arguments:

`enabled` enabled

Method `setPresentation()`**:** Set presentation

Usage:

```
GSDimension$setPresentation(presentation, interval = NULL)
```

Arguments:

`presentation` presentation. Possible values: "LIST", "CONTINUOUS_INTERVAL", "DISCRETE_INTERVAL"

`interval` interval

Method `setUnit()`**:** Set unit

Usage:

```
GSDimension$setUnit(unit)
```

Arguments:

`unit` unit

Method `setUnitSymbol()`**:** Set unit symbol

Usage:

```
GSDimension$setUnitSymbol(unitSymbol)
```

Arguments:

`unitSymbol` unit symbol

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSDimension$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
dim <- GSDimension$new()
```

`GSFeatureDimension` *A GeoServer dimension*

Description

This class models a GeoServer feature dimension.

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer feature dimension

Super classes

`geosapi::GSRESTResource` -> `geosapi::GSDimension` -> `GSFeatureDimension`

Public fields

`attribute` attribute

`endAttribute` end attribute

Methods

Public methods:

- GSFeatureDimension$new()
- GSFeatureDimension$decode()
- GSFeatureDimension$setAttribute()
- GSFeatureDimension$setEndAttribute()
- GSFeatureDimension$clone()

Method `new()`**:** Initializes an object of class GSFeatureDimension

Usage:

```
GSFeatureDimension$new(xml = NULL)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `decode()`**:** Decodes from XML

Usage:

```
GSFeatureDimension$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setAttribute()`**:** Set attribute

Usage:

```
GSFeatureDimension$setAttribute(attribute)
```

Arguments:

`attribute` attribute

Method `setEndAttribute()`**:** Set end attribute

Usage:

```
GSFeatureDimension$setEndAttribute(endAttribute)
```

Arguments:

`endAttribute` end attribute

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSFeatureDimension$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
dim <- GSFeatureDimension$new()
```

GSFeatureType *A GeoServer feature type*

Description

This class models a GeoServer feature type. This class is to be used for manipulating representations of vector data with GeoServer.

Format

R6Class object.

Details

Geoserver REST API Resource

Value

Object of R6Class for modelling a GeoServer feature type

Super classes

geosapi::GSRESTResource -> geosapi::GSResource -> GSFeatureType

Public fields

cqlFilter CQL filter

Methods

Public methods:

- GSFeatureType$new()
- GSFeatureType$decode()
- GSFeatureType$setCqlFilter()
- GSFeatureType$setVirtualTable()
- GSFeatureType$delVirtualTable()
- GSFeatureType$clone()

Method new(): Initializes an object of class GSFeatureType

Usage:

```
GSFeatureType$new(xml = NULL)
```

Arguments:

xml object of class XMLInternalNode-class

Method decode(): Decodes from XML

Usage:

```
GSFeatureType$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setCqlFilter()`**:** Set CQL filter

Usage:

```
GSFeatureType$setCqlFilter(cqlFilter)
```

Arguments:

`cqlFilter` CQL filter

Method `setVirtualTable()`**:** Set virtual table

Usage:

```
GSFeatureType$setVirtualTable(vt)
```

Arguments:

`vt` object of class GSVirtualTable

Returns: `TRUE` if set/added, `FALSE` otherwise

Method `delVirtualTable()`**:** Deletes virtual table

Usage:

```
GSFeatureType$delVirtualTable()
```

Arguments:

`vt` object of class GSVirtualTable

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSFeatureType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
ft <- GSFeatureType$new()
```

GSGeoPackageDataStore *Geoserver REST API GeoPackageDataStore*

Description

Geoserver REST API GeoPackageDataStore

Geoserver REST API GeoPackageDataStore

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer GeoPackage dataStore

Methods inherited from GSAbstractDBDataStore

`setDatabaseType(dbtype)` Sets the database type, here "geopkg

`setNamespace(namespace)` Sets the datastore namespace

`setHost(host)` Sets the database host

`setPort(port)` Set the database port

`setDatabase(database)` Set the database name

`setSchema(schema)` Set the database schema

`setUser(user)` Set the database username

`setPassword(password)` Set the database password

`setJndiReferenceName(jndiReferenceName)` Set a JNDI reference name

`setExposePrimaryKeys(exposePrimaryKeys)` Set TRUE if primary keys have to be exposed to datastore, FALSE otherwise.

`setMaxConnections(maxConnections)` Set the maximum number of connections. Default is set to 10.

`setMinConnections(minConnections)` Set the minimum number of connections. Default is set to 1.

`setFetchSize(fetchSize)` Set the fetch size. Default is set to 10.

`setConnectionTimeout(seconds)` Set the connection timeout. Default is set to 20s.

`setValidateConnections(validateConnections)` Set TRUE if connections have to be validated, FALSE otherwise.

`setPrimaryKeyMetadataTable(primaryKeyMetadataTable)` Set the name of the primaryKey metadata table

`setLooseBBox(looseBBox)` Set losse bbox parameter.

`setPreparedStatements(preparedStatements)` Set prepared statements

`setMaxOpenPreparedStatements(maxOpenPreparedStatements)` Set maximum open prepared statements

`setEstimatedExtends(estimatedExtends)` Set estimatedExtend parameter

`setDefautConnectionParameters()` Set default connection parameters

Methods

`new(xml, name, description, enabled, database)` Instantiates a GSGeoPackageDataStore object

Super classes

`geosapi::GSRESTResource` -> `geosapi::GSAbstractStore` -> `geosapi::GSAbstractDataStore` -> `geosapi::GSAbstractDBDataStore` -> `GSGeoPackageDataStore`

Methods

Public methods:

- `GSGeoPackageDataStore$new()`
- `GSGeoPackageDataStore$clone()`

Method `new()`**:** initializes an GeoPackage data store

Usage:

```
GSGeoPackageDataStore$new(
  xml = NULL,
  name = NULL,
  description = "",
  enabled = TRUE,
  database = NULL
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`name` coverage store name

`description` coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

`database` database

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSGeoPackageDataStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
ds <- GSGeoPackageDataStore$new(
 name = "ds", description = "des",
 enabled = TRUE, database = NULL
)
```

GSGeoTIFFCoverageStore

Geoserver REST API GeoTIFF CoverageStore

Description

Geoserver REST API GeoTIFF CoverageStore

Geoserver REST API GeoTIFF CoverageStore

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer GeoTIFF CoverageStore

Super classes

geosapi::GSRESTResource -> geosapi::GSAbstractStore -> geosapi::GSAbstractCoverageStore -> GSGeoTIFFCoverageStore

Public fields

`url` url

Methods

Public methods:

- GSGeoTIFFCoverageStore$new()
- GSGeoTIFFCoverageStore$clone()

Method `new()`**:** Initializes an GeoTIFF coverage store

Usage:

```
GSGeoTIFFCoverageStore$new(
  xml = NULL,
  name = NULL,
  description = "",
  enabled = TRUE,
  url = NULL
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`name` coverage store name

`description` coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

`url` url

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSGeoTIFFCoverageStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`GSImageMosaicCoverageStore`

Geoserver REST API ImageMosaicCoverageStore

Description

Geoserver REST API ImageMosaicCoverageStore

Geoserver REST API ImageMosaicCoverageStore

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer ImageMosaic CoverageStore

Super classes

`geosapi::GSRESTResource` -> `geosapi::GSAbstractStore` -> `geosapi::GSAbstractCoverageStore` -> `GSImageMosaicCoverageStore`

Public fields

`url` url

Methods

Public methods:

- `GSImageMosaicCoverageStore$new()`
- `GSImageMosaicCoverageStore$clone()`

Method `new()`**:** Initializes an Image Mosaic coverage store

Usage:

```
GSImageMosaicCoverageStore$new(
  xml = NULL,
  name = NULL,
  description = "",
  enabled = TRUE,
  url = NULL
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`name` coverage store name

`description` coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

`url` url

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSImageMosaicCoverageStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`GSInputCoverageBand` *Geoserver REST API GSInputCoverageBand*

Description

Geoserver REST API GSInputCoverageBand

Geoserver REST API GSInputCoverageBand

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer input coverage band

Methods

`new(xml, coverageName, band)` This method is used to instantiate a `GSInputCoverageBand`

`decode(xml)` This method is used to decode a `GSInputCoverageBand` from XML

`setCoverageName(coverageName)` Sets the coverage name

`setBand(band)` Sets the coverage band

Super class

`geosapi::GSRESTResource` -> `GSInputCoverageBand`

Public fields

`coverageName` coverage name

`band` band

Methods

Public methods:

- `GSInputCoverageBand$new()`
- `GSInputCoverageBand$decode()`
- `GSInputCoverageBand$setCoverageName()`
- `GSInputCoverageBand$setBand()`
- `GSInputCoverageBand$clone()`

Method `new()`**:** Initializes an object of class GSInputCoverageBand

Usage:

```
GSInputCoverageBand$new(xml = NULL, coverageName = NULL, band = NULL)
```

Arguments:

`xml` object of class XMLInternalNode-class

`coverageName` coverage name

`band` band name

Method `decode()`**:** Decodes from XML

Usage:

```
GSInputCoverageBand$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setCoverageName()`: Set coverage name

Usage:

```
GSInputCoverageBand$setCoverageName(coverageName)
```

Arguments:

`coverageName` coverage name

Method `setBand()`: Set band

Usage:

```
GSInputCoverageBand$setBand(band)
```

Arguments:

`band` band

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
GSInputCoverageBand$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSInputCoverageBand$new()
```

`GSLayer` *A GeoServer layer resource*

Description

This class models a GeoServer layer. This class is to be used for published resource (feature type or coverage).

This class models a GeoServer style.

Format

`R6Class` object.

`R6Class` object.

Details

Geoserver REST API Resource

Geoserver REST API Style

Value

Object of R6Class for modelling a GeoServer layer

Object of R6Class for modelling a GeoServer style

Super class

geosapi::GSRESTResource -> GSLayer

Public fields

full full

name name

path path

defaultStyle default style

styles styles

enabled enabled

queryable queryable

advertised advertised

Methods

Public methods:

- GSLayer$new()
- GSLayer$decode()
- GSLayer$setName()
- GSLayer$setPath()
- GSLayer$setEnabled()
- GSLayer$setQueryable()
- GSLayer$setAdvertised()
- GSLayer$setDefaultStyle()
- GSLayer$setStyles()
- GSLayer$addStyle()
- GSLayer$delStyle()
- GSLayer$clone()

Method new(): Initializes an object of class GSLayer

Usage:

```
GSLayer$new(xml = NULL)
```

Arguments:

xml object of class XMLInternalNode-class

Method decode(): Decodes from XML

Usage:

```
GSLayer$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setName()`: Set name

Usage:

```
GSLayer$setName(name)
```

Arguments:

`name` name

Method `setPath()`: Set path

Usage:

```
GSLayer$setPath(path)
```

Arguments:

`path` path

Method `setEnabled()`: Set enabled

Usage:

```
GSLayer$setEnabled(enabled)
```

Arguments:

`enabled` enabled

Method `setQueryable()`: Set queryable

Usage:

```
GSLayer$setQueryable(queryable)
```

Arguments:

`queryable` queryable

Method `setAdvertised()`: Set advertised

Usage:

```
GSLayer$setAdvertised(advertised)
```

Arguments:

`advertised` advertised

Method `setDefaultStyle()`: Set default style

Usage:

```
GSLayer$setDefaultStyle(style)
```

Arguments:

`style` object o class GSStyle or `character`

Method `setStyles()`: Set styles

Usage:

```
GSLayer$setStyles(styles)
```

Arguments:

`styles` styles

Method `addStyle()`**:** Adds style

Usage:

```
GSLayer$addStyle(style)
```

Arguments:

`style` style, object o class GSStyle or `character`

Returns: `TRUE` if added, `FALSE` otherwise

Method `delStyle()`**:** Deletes style

Usage:

```
GSLayer$delStyle(style)
```

Arguments:

`style` style, object o class GSStyle or `character`

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSLayer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Super class

`geosapi::GSRESTResource` -> `GSStyle`

Public fields

`full` full

`name` name

`filename` filename

Methods

Public methods:

- `GSStyle$new()`
- `GSStyle$decode()`
- `GSStyle$setName()`
- `GSStyle$setFilename()`
- `GSStyle$clone()`

Method `new()`**:** Initializes a GSStyle

Usage:

```
GSStyle$new(xml = NULL, name = NULL, filename = NULL)
```

Arguments:

`xml` an object of class XMLInternalNode-class

`name` name

`filename` filename

Method `decode()`**:** Decodes from XML

Usage:

```
GSStyle$decode(xml)
```

Arguments:

`xml` an object of class XMLInternalNode-class

Method `setName()`**:** set name

Usage:

```
GSStyle$setName(name)
```

Arguments:

`name` name

Method `setFilename()`**:** Set filename

Usage:

```
GSStyle$setFilename(filename)
```

Arguments:

`filename` filename

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSStyle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
lyr <- GSLayer$new()

lyr <- GSStyle$new()
```

GSLayerGroup *A GeoServer layergroup resource*

Description

This class models a GeoServer layer group. This class is to be used for clustering layers into a group.

Format

R6Class object.

Details

Geoserver REST API LayerGroup

Value

Object of R6Class for modelling a GeoServer layergroup

Super class

geosapi::GSRESTResource -> GSLayerGroup

Public fields

full full

name name

mode mode

title title

abstractTxt abstract

workspace workspace

publishables publishables

styles styles

metadataLinks metadata links

bounds bounds

Methods

Public methods:

- GSLayerGroup$new()
- GSLayerGroup$decode()
- GSLayerGroup$setName()
- GSLayerGroup$setMode()

- GSLayerGroup$setTitle()
- GSLayerGroup$setAbstract()
- GSLayerGroup$setWorkspace()
- GSLayerGroup$addLayer()
- GSLayerGroup$addLayerGroup()
- GSLayerGroup$addPublishable()
- GSLayerGroup$setStyles()
- GSLayerGroup$addStyle()
- GSLayerGroup$setMetadataLinks()
- GSLayerGroup$addMetadataLink()
- GSLayerGroup$deleteMetadataLink()
- GSLayerGroup$setBounds()
- GSLayerGroup$clone()

Method `new()`**:** Initializes an object of class GSLayerGroup

Usage:

```
GSLayerGroup$new(xml = NULL)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `decode()`**:** Decodes from XML

Usage:

```
GSLayerGroup$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setName()`**:** Set name

Usage:

```
GSLayerGroup$setName(name)
```

Arguments:

`name` name

Method `setMode()`**:** Set mode

Usage:

```
GSLayerGroup$setMode(mode)
```

Arguments:

`mode` a mode value among "SINGLE", "NAMED", "CONTAINER", "EO"

Method `setTitle()`**:** Set title

Usage:

```
GSLayerGroup$setTitle(title)
```

Arguments:

`title` title

Method `setAbstract()`: Set abstract

Usage:

```
GSLayerGroup$setAbstract(abstract)
```

Arguments:

`abstract` abstract

Method `setWorkspace()`: Set workspace

Usage:

```
GSLayerGroup$setWorkspace(workspace)
```

Arguments:

`workspace` workspace name, object of class GSWorkspace or `character`

Method `addLayer()`: Adds layer

Usage:

```
GSLayerGroup$addLayer(layer, style)
```

Arguments:

`layer` layer name
`style` style name

Method `addLayerGroup()`: Adds layer group

Usage:

```
GSLayerGroup$addLayerGroup(layerGroup)
```

Arguments:

`layerGroup` layer group

Method `addPublishable()`: Adds publishable

Usage:

```
GSLayerGroup$addPublishable(publishable)
```

Arguments:

`publishable` publishable

Returns: `TRUE` if added, `FALSE` otherwise

Method `setStyles()`: Set styles

Usage:

```
GSLayerGroup$setStyles(styles)
```

Arguments:

`styles` styles

Method `addStyle()`: Adds a style

Usage:

```
GSLayerGroup$addStyle(style)
```

Arguments:

`style` style

Returns: `TRUE` if added, `FALSE` otherwise

Method `setMetadataLinks()`**:** Set metadata links

Usage:

```
GSLayerGroup$setMetadataLinks(metadataLinks)
```

Arguments:

`metadataLinks` metadata links

Method `addMetadataLink()`**:** Adds metadata link

Usage:

```
GSLayerGroup$addMetadataLink(metadataLink)
```

Arguments:

`metadataLink` object of class GSMetadataLink

Returns: `TRUE` if added, `FALSE` otherwise

Method `deleteMetadataLink()`**:** Deletes metadata link

Usage:

```
GSLayerGroup$deleteMetadataLink(metadataLink)
```

Arguments:

`metadataLink` object of class GSMetadataLink

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `setBounds()`**:** Set bounds

Usage:

```
GSLayerGroup$setBounds(minx, miny, maxx, maxy, bbox = NULL, crs)
```

Arguments:

`minx` minx

`miny` miny

`maxx` maxx

`maxy` maxy

`bbox` bbox

`crs` crs

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSLayerGroup$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
lyr <- GSLayerGroup$new()
```

GSLayerManager *Geoserver REST API Layer Manager*

Description

Geoserver REST API Layer Manager

Geoserver REST API Layer Manager

Format

R6Class object.

Value

Object of R6Class with methods for managing GeoServer Layers as results of published feature types or coverages

Super class

geosapi::GSManager -> GSLayerManager

Methods

Public methods:

- GSLayerManager$getLayers()
- GSLayerManager$getLayerNames()
- GSLayerManager$getLayer()
- GSLayerManager$createLayer()
- GSLayerManager$updateLayer()
- GSLayerManager$deleteLayer()
- GSLayerManager$getLayerGroups()
- GSLayerManager$getLayerGroupNames()
- GSLayerManager$getLayerGroup()
- GSLayerManager$createLayerGroup()
- GSLayerManager$updateLayerGroup()
- GSLayerManager$deleteLayerGroup()
- GSLayerManager$clone()

Method `getLayers()`**:** Get the list of layers.

Usage:

```
GSLayerManager$getLayers()
```

Returns: an object of class `list` giving items of class `GSLayer`

Method `getLayerNames()`**:** Get the list of layer names.

Usage:

```
GSLayerManager$getLayerNames()
```

Returns: a vector of class `character`

Method `getLayer()`**:** Get layer by name

Usage:

```
GSLayerManager$getLayer(lyr)
```

Arguments:

`lyr` layer name

Returns: an object of class GSLayer

Method `createLayer()`**:** Creates a new layer given an object of class `GSLayer`

Usage:

```
GSLayerManager$createLayer(layer)
```

Arguments:

`layer` object of class GSLayer

Returns: `TRUE` if created, `FALSE` otherwise

Method `updateLayer()`**:** Updates a layer given an object of class `GSLayer`

Usage:

```
GSLayerManager$updateLayer(layer)
```

Arguments:

`layer` object of class GSLayer

Returns: `TRUE` if updated, `FALSE` otherwise

Method `deleteLayer()`**:** Deletes layer given an object of class `GSLayer`

Usage:

```
GSLayerManager$deleteLayer(lyr)
```

Arguments:

`lyr` layer name

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `getLayerGroups()`**:** Get layer groups

Usage:

```
GSLayerManager$getLayerGroups(ws = NULL)
```

Arguments:

`ws` workspace name. Optional

Returns: a list of objects of class GSLayerGroup

Method `getLayerGroupNames()`**:** Get layer group names

Usage:

```
GSLayerManager$getLayerGroupNames(ws = NULL)
```

Arguments:

`ws` workspace name

Returns: a list of layer group names, as vector of class `character`

Method `getLayerGroup()`**:** Get layer group

Usage:

```
GSLayerManager$getLayerGroup(lyr, ws = NULL)
```

Arguments:

`lyr` lyr

`ws` workspace name

Returns: an object of class GSLayerGroup

Method `createLayerGroup()`**:** Creates a layer group

Usage:

```
GSLayerManager$createLayerGroup(layerGroup, ws = NULL)
```

Arguments:

`layerGroup` object of class GSLayerGroup

`ws` workspace name. Optional

Returns: `TRUE` if created, `FALSE` otherwise

Method `updateLayerGroup()`**:** Updates a layer group

Usage:

```
GSLayerManager$updateLayerGroup(layerGroup, ws = NULL)
```

Arguments:

`layerGroup` object of class GSLayerGroup

`ws` workspace name. Optional

Returns: `TRUE` if updated, `FALSE` otherwise

Method `deleteLayerGroup()`**:** Deletes a layer group

Usage:

```
GSLayerManager$deleteLayerGroup(lyr, ws = NULL)
```

Arguments:

`lyr` layer group name

`ws` workspace name. Optional

Returns: TRUE if deleted, FALSE otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSLayerManager$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
## Not run:
   GSLayerManager$new("http://localhost:8080/geoserver", "admin", "geoserver")

## End(Not run)
```

`GSManager` *Geoserver REST API Manager*

Description

Geoserver REST API Manager

Geoserver REST API Manager

Format

R6Class object.

Value

Object of R6Class with methods for communication with the REST API of a GeoServer instance.

Methods

`new(url, user, pwd, logger, keyring_backend)`

`logger(type, text)` Basic logger to report geosapi logs. Used internally

`INFO(text)` Logger to report information. Used internally

`WARN(text)` Logger to report warnings. Used internally

`ERROR(text)` Logger to report errors. Used internally

`getUrl()` Get the authentication URL

`connect()` This methods attempts a connection to GeoServer REST API. User internally during initialization of `GSManager`.

`reload()` Reloads the GeoServer catalog.

`getSystemStatus()` Get system status

`getClassName()` Retrieves the name of the class instance

`getWorkspaceManager()` Retrieves an instance of workspace manager

`getNamespaceManager()` Retrieves an instance of namespace manager

`getDataStoreManager()` Retrieves an instance of datastore manager

Public fields

`verbose.info` if geosapi logs have to be printed

`verbose.debug` if curl logs have to be printed

`loggerType` the type of logger

`url` the Base url of GeoServer

`version` the version of Geoserver. Handled as `GSVersion` object

Methods

Public methods:

- `GSManager$logger()`
- `GSManager$INFO()`
- `GSManager$WARN()`
- `GSManager$ERROR()`
- `GSManager$new()`
- `GSManager$getUrl()`
- `GSManager$connect()`
- `GSManager$reload()`
- `GSManager$getSystemStatus()`
- `GSManager$monitor()`
- `GSManager$getClassName()`
- `GSManager$getWorkspaceManager()`
- `GSManager$getNamespaceManager()`
- `GSManager$getDataStoreManager()`
- `GSManager$getCoverageStoreManager()`
- `GSManager$getServiceManager()`
- `GSManager$getStyleManager()`
- `GSManager$clone()`

Method `logger()`**:** Prints a log message

Usage:

```
GSManager$logger(type, text)
```

Arguments:

`type` type of log, "INFO", "WARN", "ERROR"

text text

Method INFO(): Prints an INFO log message

Usage:

```
GSManager$INFO(text)
```

Arguments:

text text

Method WARN(): Prints an WARN log message

Usage:

```
GSManager$WARN(text)
```

Arguments:

text text

Method ERROR(): Prints an ERROR log message

Usage:

```
GSManager$ERROR(text)
```

Arguments:

text text

Method new(): This method is used to instantiate a GSManager with the url of the GeoServer and credentials to authenticate (user/pwd).

By default, the logger argument will be set to NULL (no logger). This argument accepts two possible values: INFO: to print only geosapi logs, DEBUG: to print geosapi and CURL logs.

The keyring_backend can be set to use a different backend for storing the Geoserver user password with **keyring** (Default value is 'env').

Usage:

```
GSManager$new(url, user, pwd, logger = NULL, keyring_backend = "env")
```

Arguments:

url url

user user

pwd pwd

logger logger

keyring_backend keyring backend. Default is 'env'

Method getUrl(): Get URL

Usage:

```
GSManager$getUrl()
```

Returns: the Geoserver URL

Method connect(): Connects to geoServer

Usage:

```
GSManager$connect()
```

Returns: TRUE if connected, raises an error otherwise

Method `reload()`**:** Reloads the GeoServer catalog

Usage:

```
GSManager$reload()
```

Returns: TRUE if reloaded, FALSE otherwise

Method `getSystemStatus()`**:** Get system status

Usage:

```
GSManager$getSystemStatus()
```

Returns: an object of class `data.frame` given the date time and metrics value

Method `monitor()`**:** Monitors the Geoserver by launching a small shiny monitoring application

Usage:

```
GSManager$monitor(file = NULL, append = FALSE, sleep = 1)
```

Arguments:

`file` file where to store monitoring results
`append` whether to append results to existing files
`sleep` sleeping interval to trigger a system status call

Method `getClassName()`**:** Get class name

Usage:

```
GSManager$getClassName()
```

Returns: the self class name, as `character`

Method `getWorkspaceManager()`**:** Get Workspace manager

Usage:

```
GSManager$getWorkspaceManager()
```

Returns: an object of class GSWorkspaceManager

Method `getNamespaceManager()`**:** Get Namespace manager

Usage:

```
GSManager$getNamespaceManager()
```

Returns: an object of class GSNamespaceManager

Method `getDataStoreManager()`**:** Get Datastore manager

Usage:

```
GSManager$getDataStoreManager()
```

Returns: an object of class GSDataStoreManager

Method `getCoverageStoreManager()`**:** Get Coverage store manager

Usage:

```
GSManager$getCoverageStoreManager()
```

Returns: an object of class GSCoverageStoreManager

Method `getServiceManager()`**:** Get service manager

Usage:

```
GSManager$getServiceManager()
```

Returns: an object of class GSServiceManager

Method `getStyleManager()`**:** Get style manager

Usage:

```
GSManager$getStyleManager()
```

Returns: an object of class GSStyleManager

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSManager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
## Not run:
   GSManager$new("http://localhost:8080/geoserver", "admin", "geoserver")

## End(Not run)
```

`GSMetadataLink` *A GeoServer resource metadataLink*

Description

This class models a GeoServer resource metadataLink made of a type (free text e.g. text/xml, text/html), a metadataType (Possible values are ISO19115:2003, FGDC, TC211, 19139, other), and a content: an URL that gives the metadataLink

Format

`R6Class` object.

Details

Geoserver REST API Metadatalink

Value

Object of R6Class for modelling a GeoServer resource metadataLink

Super class

geosapi::GSRESTResource -> GSMetadataLink

Public fields

type type

metadataType metadata type

content content

Methods

Public methods:

- GSMetadataLink$new()
- GSMetadataLink$decode()
- GSMetadataLink$setType()
- GSMetadataLink$setMetadataType()
- GSMetadataLink$setContent()
- GSMetadataLink$clone()

Method new(): Initializes an object of class GSMetadataLink

Usage:

```
GSMetadataLink$new(xml = NULL, type, metadataType, content)
```

Arguments:

xml object of class XMLInternalNode-class

type type

metadataType metadata type

content content

Method decode(): Decodes from XML

Usage:

```
GSMetadataLink$decode(xml)
```

Arguments:

xml object of class XMLInternalNode-class

Method setType(): Set type type

Usage:

```
GSMetadataLink$setType(type)
```

Arguments:

type type

Method `setMetadataType()`**:** Set metadata type

Usage:

```
GSMetadataLink$setMetadataType(metadataType)
```

Arguments:

`metadataType` metadata type. Supported values: "ISO19115:2003", "FGDC", "TC211", "19139", "other"

Method `setContent()`**:** Set content

Usage:

```
GSMetadataLink$setContent(content)
```

Arguments:

`content` content

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSMetadataLink$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`GSNamespace` *Geoserver REST API Namespace*

Description

Geoserver REST API Namespace

Geoserver REST API Namespace

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer namespace

Super class

`geosapi::GSRESTResource` -> `GSNamespace`

Public fields

`name` namespace name

`prefix` namespace prefix

`uri` namespace URI

`full` completeness of the namespace description

Methods

Public methods:

- `GSNamespace$new()`
- `GSNamespace$decode()`
- `GSNamespace$clone()`

Method `new()`**:** Initializes an object of class GSNamespace

Usage:

```
GSNamespace$new(xml = NULL, prefix, uri)
```

Arguments:

`xml` object of class XMLInternalNode-class

`prefix` prefix

`uri` uri

Method `decode()`**:** Decodes from XML

Usage:

```
GSNamespace$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSNamespace$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSNamespace$new(prefix = "prefix", uri = "http://prefix")
```

GSNamespaceManager *Geoserver REST API Namespace Manager*

Description

Geoserver REST API Namespace Manager

Geoserver REST API Namespace Manager

Format

R6Class object.

Value

Object of R6Class with methods for managing the namespaces of a GeoServer instance.

Super class

geosapi::GSManager -> GSNamespaceManager

Methods

Public methods:

- GSNamespaceManager$getNamespaces()
- GSNamespaceManager$getNamespaceNames()
- GSNamespaceManager$getNamespace()
- GSNamespaceManager$createNamespace()
- GSNamespaceManager$updateNamespace()
- GSNamespaceManager$deleteNamespace()
- GSNamespaceManager$clone()

Method `getNamespaces()`**:** Get the list of available namespace. Re

Usage:

```
GSNamespaceManager$getNamespaces()
```

Returns: an object of class `list` containing items of class GSNamespace

Method `getNamespaceNames()`**:** Get the list of available namespace names.

Usage:

```
GSNamespaceManager$getNamespaceNames()
```

Returns: a vector of class `character`

Method `getNamespace()`**:** Get a GSNamespace object given a namespace name.

Usage:

```
GSNamespaceManager$getNamespace(ns)
```

Arguments:

`ns` namespace

Returns: an object of class GSNamespace

Method `createNamespace()`**:** Creates a GeoServer namespace given a prefix, and an optional URI.

Usage:

```
GSNamespaceManager$createNamespace(prefix, uri)
```

Arguments:

`prefix` prefix

`uri` uri

Returns: `TRUE` if the namespace has been successfully created, `FALSE` otherwise

Method `updateNamespace()`**:** Updates a GeoServer namespace given a prefix, and an optional URI.

Usage:

```
GSNamespaceManager$updateNamespace(prefix, uri)
```

Arguments:

`prefix` prefix

`uri` uri

Returns: `TRUE` if the namespace has been successfully updated, `FALSE` otherwise

Method `deleteNamespace()`**:** Deletes a GeoServer namespace given a name.

Usage:

```
GSNamespaceManager$deleteNamespace(name, recurse = FALSE)
```

Arguments:

`name` name

`recurse` recurse

Returns: `TRUE` if the namespace has been successfully deleted, `FALSE` otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSNamespaceManager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
## Not run:
   GSNamespaceManager$new("http://localhost:8080/geoserver", "admin", "geoserver")

## End(Not run)
```

GSOracleNGDataStore *Geoserver REST API OracleNGDataStore*

Description

Geoserver REST API OracleNGDataStore

Geoserver REST API OracleNGDataStore

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer OracleNG dataStore

Methods inherited from GSAbstractDBDataStore

`setDatabaseType(dbtype)` Sets the database type, here "OracleNG"

`setNamespace(namespace)` Sets the datastore namespace

`setHost(host)` Sets the database host

`setPort(port)` Set the database port

`setDatabase(database)` Set the database name

`setSchema(schema)` Set the database schema

`setUser(user)` Set the database username

`setPassword(password)` Set the database password

`setJndiReferenceName(jndiReferenceName)` Set a JNDI reference name

`setExposePrimaryKeys(exposePrimaryKeys)` Set TRUE if primary keys have to be exposed to datastore, FALSE otherwise.

`setMaxConnections(maxConnections)` Set the maximum number of connections. Default is set to 10.

`setMinConnections(minConnections)` Set the minimum number of connections. Default is set to 1.

`setFetchSize(fetchSize)` Set the fetch size. Default is set to 10.

`setConnectionTimeout(seconds)` Set the connection timeout. Default is set to 20s.

`setValidateConnections(validateConnections)` Set TRUE if connections have to be validated, FALSE otherwise.

`setPrimaryKeyMetadataTable(primaryKeyMetadataTable)` Set the name of the primaryKey metadata table

`setLooseBBox(looseBBox)` Set losse bbox parameter.

`setPreparedStatements(preparedStatements)` Set prepared statements

`setMaxOpenPreparedStatements(maxOpenPreparedStatements)` Set maximum open prepared statements

`setEstimatedExtends(estimatedExtends)` Set estimatedExtend parameter

`setDefautConnectionParameters()` Set default connection parameters

Methods

`new(xml, name, description, enabled)` Instantiates a GSOracleNGDataStore object

Super classes

`geosapi::GSRESTResource` -> `geosapi::GSAbstractStore` -> `geosapi::GSAbstractDataStore` -> `geosapi::GSAbstractDBDataStore` -> `GSOracleNGDataStore`

Methods

Public methods:

- `GSOracleNGDataStore$new()`
- `GSOracleNGDataStore$clone()`

Method `new()`**:** initializes an Oracle NG data store

Usage:

```
GSOracleNGDataStore$new(
  xml = NULL,
  name = NULL,
  description = "",
  enabled = TRUE
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`name` coverage store name

`description` coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSOracleNGDataStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSOracleNGDataStore$new(name = "ds", description = "des", enabled = TRUE)
```

GSPostGISDataStore *Geoserver REST API PostGISDataStore*

Description

Geoserver REST API PostGISDataStore

Geoserver REST API PostGISDataStore

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer PostGIS dataStore

Super classes

geosapi::GSRESTResource -> geosapi::GSAbstractStore -> geosapi::GSAbstractDataStore -> geosapi::GSAbstractDBDataStore -> GSPostGISDataStore

Methods

Public methods:

- GSPostGISDataStore$new()
- GSPostGISDataStore$clone()

Method `new()`**:** initializes a PostGIS data store

Usage:

```
GSPostGISDataStore$new(
  xml = NULL,
  name = NULL,
  description = "",
  enabled = TRUE
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`name` coverage store name

`description` coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSPostGISDataStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSPostGISDataStore$new(name = "ds", description = "des", enabled = TRUE)
```

GSPublishable *A GeoServer layer group publishable*

Description

This class models a GeoServer layer. This class is to be used internally by **geosapi** for configuring layers or layer groups within an object of class `GSLayerGroup`

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer layer group publishable

Super class

`geosapi::GSRESTResource` -> `GSPublishable`

Public fields

`full` full

`name` name

`attr_type` type of attribute

Methods

Public methods:

- `GSPublishable$new()`
- `GSPublishable$decode()`
- `GSPublishable$setName()`
- `GSPublishable$setType()`
- `GSPublishable$clone()`

Method `new()`**:** Initializes a GSPublishable

Usage:

```
GSPublishable$new(xml = NULL, name, type)
```

Arguments:

`xml` an object of class XMLInternalNode-class

`name` name

`type` type

Method `decode()`**:** Decodes from XML

Usage:

```
GSPublishable$decode(xml)
```

Arguments:

`xml` an object of class XMLInternalNode-class

Method `setName()`**:** set name

Usage:

```
GSPublishable$setName(name)
```

Arguments:

`name` name

Method `setType()`**:** Set type

Usage:

```
GSPublishable$setType(type)
```

Arguments:

`type` type

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSPublishable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
publishable <- GSPublishable$new(name = "name", type = "layer")
```

GSResource *A GeoServer abstract resource*

Description

This class models an abstract GeoServer resource. This class is used internally for modelling instances of class GSFeatureType or GSCoverage

Format

R6Class object.

Details

Geoserver REST API Resource

Value

Object of R6Class for modelling a GeoServer resource

Methods

new(rootName, xml) This method is used to instantiate a GSResource

decode(xml) This method is used to decode a GSResource from XML

encode() This method is used to encode a GSResource to XML. Inherited from the generic GSRESTResource encoder

setEnabled(enabled) Sets if the resource is enabled or not in GeoServer

setName(name) Sets the resource name

setNativeName(nativeName) Sets the resource native name

setTitle(title) Sets the resource title

setDescription(description) Sets the resource description

setAbstract(abstract) Sets the resource abstract

setKeywords(keywords) Sets a list of keywords

addKeyword(keyword) Sets a keyword. Returns TRUE if set, FALSE otherwise

delKeyword(keyword) Deletes a keyword. Returns TRUE if deleted, FALSE otherwise

setMetadataLinks(metadataLinks) Sets a list of GSMetadataLinks

addMetadataLink(metadataLink) Adds a metadataLink

delMetadataLink(metadataLink) Deletes a metadataLink

setNativeCRS(nativeCRS) Sets the resource nativeCRS

setSrs(srs) Sets the resource srs

setNativeBoundingBox(minx, miny, maxx, maxy, bbox, crs) Sets the resource nativeBoundingBox. Either from coordinates or from a bbox object (matrix).

setLatLonBoundingBox(minx, miny, maxx, maxy, bbox, crs) Sets the resource latLonBoundingBox. Either from coordinates or from a bbox object (matrix).

setProjectionPolicy(policy) Sets the resource projection policy

Super class

geosapi::GSRESTResource -> GSResource

Public fields

full full

name resource name

nativeName resource native name

title resource title

description resource description

abstract resource abstract

keywords resource keywords

metadataLinks resource metadata links

nativeCRS resource native CRS

srs resource srs

nativeBoundingBox resource lat/lon native bounding box

latLonBoundingBox resource lat/lon bounding box

projectionPolicy resource projection policy

enabled enabled

metadata metadata

Methods

Public methods:

- GSResource$new()
- GSResource$decode()
- GSResource$setEnabled()
- GSResource$setName()
- GSResource$setNativeName()
- GSResource$setTitle()
- GSResource$setDescription()
- GSResource$setAbstract()
- GSResource$setKeywords()
- GSResource$addKeyword()
- GSResource$delKeyword()
- GSResource$setMetadataLinks()
- GSResource$addMetadataLink()
- GSResource$deleteMetadataLink()
- GSResource$setProjectionPolicy()
- GSResource$setSrs()
- GSResource$setNativeCRS()

- GSResource$setLatLonBoundingBox()
- GSResource$setNativeBoundingBox()
- GSResource$setMetadata()
- GSResource$delMetadata()
- GSResource$setMetadataDimension()
- GSResource$clone()

Method `new()`**:** Initializes a GSResource

Usage:

```
GSResource$new(rootName = NULL, xml = NULL)
```

Arguments:

`rootName` root name

`xml` object of class XMLInternalNode-class

Method `decode()`**:** Decodes from XML

Usage:

```
GSResource$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setEnabled()`**:** Set enabled

Usage:

```
GSResource$setEnabled(enabled)
```

Arguments:

`enabled` enabled

Method `setName()`**:** Set name

Usage:

```
GSResource$setName(name)
```

Arguments:

`name` name

Method `setNativeName()`**:** Set native name

Usage:

```
GSResource$setNativeName(nativeName)
```

Arguments:

`nativeName` native name

Method `setTitle()`**:** Set title

Usage:

```
GSResource$setTitle(title)
```

Arguments:

`title` title

Method `setDescription()`**:** Set description

Usage:

```
GSResource$setDescription(description)
```

Arguments:

`description` description

Method `setAbstract()`**:** Set abstract

Usage:

```
GSResource$setAbstract(abstract)
```

Arguments:

`abstract` abstract

Method `setKeywords()`**:** Set keyword(s)

Usage:

```
GSResource$setKeywords(keywords)
```

Arguments:

`keywords` keywords

Method `addKeyword()`**:** Adds keyword

Usage:

```
GSResource$addKeyword(keyword)
```

Arguments:

`keyword` keyword

Returns: `TRUE` if added, `FALSE` otherwise

Method `delKeyword()`**:** Deletes keyword

Usage:

```
GSResource$delKeyword(keyword)
```

Arguments:

`keyword` keyword

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `setMetadataLinks()`**:** Set metadata links

Usage:

```
GSResource$setMetadataLinks(metadataLinks)
```

Arguments:

`metadataLinks` metadata links

Method `addMetadataLink()`**:** Adds metadata link

Usage:

```
GSResource$addMetadataLink(metadataLink)
```

Arguments:

`metadataLink` object of class GSMetadataLink

Returns: `TRUE` if added, `FALSE` otherwise

Method `deleteMetadataLink()`**:** Deletes metadata link

Usage:

```
GSResource$deleteMetadataLink(metadataLink)
```

Arguments:

`metadataLink` object of class GSMetadataLink

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `setProjectionPolicy()`**:** Set projection policy

Usage:

```
GSResource$setProjectionPolicy(projectionPolicy)
```

Arguments:

`projectionPolicy` projection policy

Method `setSrs()`**:** Set SRS

Usage:

```
GSResource$setSrs(srs)
```

Arguments:

`srs` srs

Method `setNativeCRS()`**:** Set native CRS

Usage:

```
GSResource$setNativeCRS(nativeCRS)
```

Arguments:

`nativeCRS` native crs

Method `setLatLonBoundingBox()`**:** Set LatLon bounding box

Usage:

```
GSResource$setLatLonBoundingBox(minx, miny, maxx, maxy, bbox = NULL, crs)
```

Arguments:

`minx` minx
`miny` miny
`maxx` maxx
`maxy` maxy
`bbox` bbox
`crs` crs

Method `setNativeBoundingBox()`**:** Set native bounding box

Usage:

```
GSResource$setNativeBoundingBox(minx, miny, maxx, maxy, bbox = NULL, crs)
```

Arguments:

`minx` minx
`miny` miny
`maxx` maxx
`maxy` maxy
`bbox` bbox
`crs` crs

Method `setMetadata()`**:** Set metadata

Usage:

```
GSResource$setMetadata(key, metadata)
```

Arguments:

`key` key
`metadata` metadata

Returns: `TRUE` if added, `FALSE` otherwise

Method `delMetadata()`**:** Deletes metadata

Usage:

```
GSResource$delMetadata(key)
```

Arguments:

`key` key

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `setMetadataDimension()`**:** Set metadata dimension

Usage:

```
GSResource$setMetadataDimension(key, dimension, custom = FALSE)
```

Arguments:

`key` key
`dimension` dimension
`custom` custom

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSResource$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
res <- GSResource$new(rootName = "featureType")
```

GSRESTEntrySet *Geoserver REST API XML entry set*

Description

Geoserver REST API XML entry set

Geoserver REST API XML entry set

Format

R6Class object.

Value

Object of R6Class for modelling a entry set

Super class

geosapi::GSRESTResource -> GSRESTEntrySet

Public fields

entryset entryset

Methods

Public methods:

- GSRESTEntrySet$new()
- GSRESTEntrySet$decode()
- GSRESTEntrySet$setEntryset()
- GSRESTEntrySet$addEntry()
- GSRESTEntrySet$setEntry()
- GSRESTEntrySet$delEntry()
- GSRESTEntrySet$clone()

Method new(): Initializes an object of class GSRESTEntrySet

Usage:

```
GSRESTEntrySet$new(rootName, xml = NULL, entryset)
```

Arguments:

rootName root name

xml object of class XMLInternalNode-class

entryset entry set

Method decode(): Decodes from XML

Usage:

```
GSRESTEntrySet$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setEntryset()`**:** Set entry set

Usage:

```
GSRESTEntrySet$setEntryset(entryset)
```

Arguments:

`entryset` entry set

Method `addEntry()`**:** Adds entry set

Usage:

```
GSRESTEntrySet$addEntry(key, value)
```

Arguments:

`key` key

`value` value

Returns: `TRUE` if added, `FALSE` otherwise

Method `setEntry()`**:** Sets entry set

Usage:

```
GSRESTEntrySet$setEntry(key, value)
```

Arguments:

`key` key

`value` value

Method `delEntry()`**:** Deletes entry set

Usage:

```
GSRESTEntrySet$delEntry(key)
```

Arguments:

`key` key

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSRESTEntrySet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

GSRESTResource *Geoserver REST API REST Resource interface*

Description

Geoserver REST API REST Resource interface

Geoserver REST API REST Resource interface

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer REST resource interface

Public fields

rootName root name

Methods

Public methods:

- GSRESTResource$new()
- GSRESTResource$decode()
- GSRESTResource$encode()
- GSRESTResource$clone()

Method new(): Initializes an object of class GSRESTResource

Usage:

```
GSRESTResource$new(xml, rootName)
```

Arguments:

xml object of class XMLInternalNode-class

rootName root name

Method decode(): Decodes from XML. Abstract method to be implemented by sub-classes

Usage:

```
GSRESTResource$decode(xml)
```

Arguments:

xml object of class XMLInternalNode-class

Method encode(): Encodes as XML

Usage:

```
GSRESTResource$encode()
```

Returns: an object of class XMLInternalNode-class

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSRESTResource$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`GSServiceManager` *Geoserver REST API Service Manager*

Description

Geoserver REST API Service Manager

Geoserver REST API Service Manager

Format

`R6Class` object.

Value

Object of `R6Class` with methods for managing GeoServer services

Constructor

`new(url, user, pwd, logger)` This method is used to instantiate a GSManager with the `url` of the GeoServer and credentials to authenticate (`user`/`pwd`). By default, the `logger` argument will be set to `NULL` (no logger). This argument accepts two possible values: `INFO`: to print only geosapi logs, `DEBUG`: to print geosapi and CURL logs

`getServiceSettings(service, ws)`

`getWmsSettings(ws)` Get WMS settings. To get the WMS settings for a specific workspace, specify the workspace name as `ws` parameter, otherwise global settings are retrieved.

`getWfsSettings(ws)` Get WFS settings. To get the WFS settings for a specific workspace, specify the workspace name as `ws` parameter, otherwise global settings are retrieved.

`getWcsSettings(ws)` Get WCS settings. To get the WCS settings for a specific workspace, specify the workspace name as `ws` parameter, otherwise global settings are retrieved.

`updateServiceSettings(serviceSettings, service, ws)` Updates the service settings with an object of class `GSServiceSetting`. An optional workspace name `ws` can be specified to update service settings applying to a workspace.

`deleteServiceSettings(service, ws)` Deletes the service settings. This method is used internally by **geosapi** for disabling a service setting at workspace level.

`updateWmsSettings(serviceSettings, ws)` Updates the WMS settings with an object of class `GSServiceSetting`. An optional workspace name `ws` can be specified to update WMS settings applying to a workspace.

`updateWfsSettings(serviceSettings, ws)` Updates the WFS settings with an object of class `GSServiceSetting`. An optional workspace name `ws` can be specified to update WFS settings applying to a workspace.

`updateWcsSettings(serviceSettings, ws)` Updates the WCS settings with an object of class `GSServiceSettings`. An optional workspace name `ws` can be specified to update WCS settings applying to a workspace.

`enableWMS(ws)` Enables the WMS, either globally, or for a given workspace (optional)

`enableWFS(ws)` Enables the WFS, either globally, or for a given workspace (optional)

`enableWCS(ws)` Enables the WCS, either globally, or for a given workspace (optional)

`disableServiceSettings(service, ws)` Disables a service, either globally, or for a given workspace (optional). For a global service setting, an UPDATE operation will be applied, while for a workspace service setting, a DELETE operation is applied.

`disableWMS(ws)` Disables the WMS, either globally, or for a given workspace (optional)

`disableWFS(ws)` Disables the WFS, either globally, or for a given workspace (optional)

`disableWCS(ws)` Disables the WCS, either globally, or for a given workspace (optional)

Super class

`geosapi::GSManager` -> `GSServiceManager`

Methods

Public methods:

- `GSServiceManager$getServiceSettings()`
- `GSServiceManager$getWmsSettings()`
- `GSServiceManager$getWfsSettings()`
- `GSServiceManager$getWcsSettings()`
- `GSServiceManager$updateServiceSettings()`
- `GSServiceManager$deleteServiceSettings()`
- `GSServiceManager$updateWmsSettings()`
- `GSServiceManager$updateWfsSettings()`
- `GSServiceManager$updateWcsSettings()`
- `GSServiceManager$enableWMS()`
- `GSServiceManager$enableWFS()`
- `GSServiceManager$enableWCS()`
- `GSServiceManager$disableServiceSettings()`
- `GSServiceManager$disableWMS()`
- `GSServiceManager$disableWFS()`

- GSServiceManager$disableWCS()
- GSServiceManager$clone()

Method `getServiceSettings()`: Get the service settings. To get the service settings for a specific workspace, specify the workspace name as `ws` parameter, otherwise global settings are retrieved.

Usage:

```
GSServiceManager$getServiceSettings(service, ws = NULL)
```

Arguments:

`service` service

`ws` workspace name

Returns: an object of class GSServiceSettings

Method `getWmsSettings()`: Get WMS settings. To get the WMS settings for a specific workspace, specify the workspace name as `ws` parameter, otherwise global settings are retrieved.

Usage:

```
GSServiceManager$getWmsSettings(ws = NULL)
```

Arguments:

`ws` workspace name

Returns: an object of class GSServiceSettings

Method `getWfsSettings()`: Get WFS settings. To get the WFS settings for a specific workspace, specify the workspace name as `ws` parameter, otherwise global settings are retrieved.

Usage:

```
GSServiceManager$getWfsSettings(ws = NULL)
```

Arguments:

`ws` workspace name

Returns: an object of class GSServiceSettings

Method `getWcsSettings()`: Get WCS settings. To get the WCS settings for a specific workspace, specify the workspace name as `ws` parameter, otherwise global settings are retrieved.

Usage:

```
GSServiceManager$getWcsSettings(ws = NULL)
```

Arguments:

`ws` workspace name

Returns: an object of class GSServiceSettings

Method `updateServiceSettings()`: Updates the service settings with an object of class `GSServiceSettings`. An optional workspace name `ws` can be specified to update service settings applying to a workspace.

Usage:

```
GSServiceManager$updateServiceSettings(serviceSettings, service, ws = NULL)
```

Arguments:

`serviceSettings` serviceSettings object of class GSServiceSettings

`service` service

`ws` workspace name

Returns: `TRUE` if updated, `FALSE` otherwise

Method `deleteServiceSettings()`**:** Deletes the service settings. This method is used internally by **geosapi** for disabling a service setting at workspace level.

Usage:

```
GSServiceManager$deleteServiceSettings(service, ws = NULL)
```

Arguments:

`service` service

`ws` workspace name

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `updateWmsSettings()`**:** Updates the WMS settings with an object of class `GSServiceSettings`. An optional workspace name `ws` can be specified to update WMS settings applying to a workspace.

Usage:

```
GSServiceManager$updateWmsSettings(serviceSettings, ws = NULL)
```

Arguments:

`serviceSettings` service settings object of class GSServiceSettings

`ws` workspace name

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `updateWfsSettings()`**:** Updates the WFS settings with an object of class `GSServiceSettings`. An optional workspace name `ws` can be specified to update WFS settings applying to a workspace.

Usage:

```
GSServiceManager$updateWfsSettings(serviceSettings, ws = NULL)
```

Arguments:

`serviceSettings` service settings object of class GSServiceSettings

`ws` workspace name

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `updateWcsSettings()`**:** Updates the WCS settings with an object of class `GSServiceSettings`. An optional workspace name `ws` can be specified to update WCS settings applying to a workspace.

Usage:

```
GSServiceManager$updateWcsSettings(serviceSettings, ws = NULL)
```

Arguments:

`serviceSettings` service settings object of class GSServiceSettings

`ws` workspace name

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `enableWMS()`**:** Enables WMS service settings

Usage:

```
GSServiceManager$enableWMS(ws = NULL)
```

Arguments:

`ws` workspace name

Returns: `TRUE` if enabled, `FALSE` otherwise

Method `enableWFS()`**:** Enables WFS service settings

Usage:

```
GSServiceManager$enableWFS(ws = NULL)
```

Arguments:

`ws` workspace name

Returns: `TRUE` if enabled, `FALSE` otherwise

Method `enableWCS()`**:** Enables WCS service settings

Usage:

```
GSServiceManager$enableWCS(ws = NULL)
```

Arguments:

`ws` workspace name

Returns: `TRUE` if enabled, `FALSE` otherwise

Method `disableServiceSettings()`**:** Disables service settings

Usage:

```
GSServiceManager$disableServiceSettings(service, ws = NULL)
```

Arguments:

`service` service
`ws` workspace name

Returns: `TRUE` if disabled, `FALSE` otherwise

Method `disableWMS()`**:** Disables WMS service settings

Usage:

```
GSServiceManager$disableWMS(ws = NULL)
```

Arguments:

`ws` workspace name

Returns: `TRUE` if disabled, `FALSE` otherwise

Method `disableWFS()`**:** Disables WFS service settings

Usage:

```
GSServiceManager$disableWFS(ws = NULL)
```

Arguments:

`ws` workspace name

Returns: `TRUE` if disabled, `FALSE` otherwise

Method `disableWCS()`**:** Disables WCS service settings

Usage:

```
GSServiceManager$disableWCS(ws = NULL)
```

Arguments:

`ws` workspace name

Returns: `TRUE` if disabled, `FALSE` otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSServiceManager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
## Not run:
   GSServiceManager$new("http://localhost:8080/geoserver", "admin", "geoserver")

## End(Not run)
```

`GSServiceSettings` *A GeoServer service settings resource*

Description

This class models a GeoServer OWS service settings.

Format

`R6Class` object.

Details

Geoserver REST API Service Setting

Value

Object of `R6Class` for modelling a GeoServer OWS service setting

Methods

`new(rootName, xml)` This method is used to instantiate a `GSServiceSettings`. This settings object is required to model/manipulate an OGC service configuration, using the method `GSManager$updateServiceSetti` or derivates.

`decode(xml)` This method is used to decode a `GSServiceSettings` from XML

`encode()` This method is used to encode a `GSServiceSettings` to XML. Inherited from the generic `GSRESTResource` encoder

`setEnabled(enabled)` Sets if the service is enabled (TRUE) or not (FALSE)

`setCiteCompliant(citeCompliant)` Sets if the service is compliant with CITE (TRUE) or not (FALSE)

`setName(name)` Sets the service name

`setTitle(title)` Sets the service title

`setAbstract(abstract)` Sets the service abstract

`setMaintainer(maintainer)` Sets the service maintainer

`setKeywords(keywords)` Sets a list of keywords

`addKeyword(keyword)` Sets a keyword. Returns TRUE if set, FALSE otherwise

`delKeyword(keyword)` Deletes a keyword. Returns TRUE if deleted, FALSE otherwise

`setOnlineResource(onlineResource)` Sets the online resource

`setSchemaBaseURL(schemaBaseURL)` Sets the schema base URL. Default is http://schemas.opengis.net

`setVerbose(verbose)` Sets verbose

Super class

`geosapi::GSRESTResource` -> `GSServiceSettings`

Public fields

`enabled` is service enabled or not?

`citeCompliant` is service cite compliant?

`name` service name

`title` service title

`maintainer` service maintainer

`abstrct` service abastract

`accessConstraints` service access constraints

`fees` service fees

`keywords` services keywords

`onlineResource` service online resource

`schemaBaseURL` service schema base URL

`verbose` service verbose or not?

Methods

Public methods:

- GSServiceSettings$new()
- GSServiceSettings$decode()
- GSServiceSettings$setEnabled()
- GSServiceSettings$setCiteCompliant()
- GSServiceSettings$setName()
- GSServiceSettings$setTitle()
- GSServiceSettings$setMaintainer()
- GSServiceSettings$setAbstract()
- GSServiceSettings$setAccessConstraints()
- GSServiceSettings$setFees()
- GSServiceSettings$setKeywords()
- GSServiceSettings$addKeyword()
- GSServiceSettings$delKeyword()
- GSServiceSettings$clone()

Method `new()`**:** Initializes an object of class GSServiceSettings

Usage:

```
GSServiceSettings$new(xml = NULL, service)
```

Arguments:

`xml` object of class XMLInternalNode-class

`service` service service acronym

Method `decode()`**:** Decodes from XML

Usage:

```
GSServiceSettings$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setEnabled()`**:** Set enabled

Usage:

```
GSServiceSettings$setEnabled(enabled)
```

Arguments:

`enabled` enabled

Method `setCiteCompliant()`**:** Set cite compliant

Usage:

```
GSServiceSettings$setCiteCompliant(citeCompliant)
```

Arguments:

`citeCompliant` cite compliant

Method `setName()`: Set name

Usage:

```
GSServiceSettings$setName(name)
```

Arguments:

`name` name

Method `setTitle()`: Set title

Usage:

```
GSServiceSettings$setTitle(title)
```

Arguments:

`title` title

Method `setMaintainer()`: Set maintainer

Usage:

```
GSServiceSettings$setMaintainer(maintainer)
```

Arguments:

`maintainer` maintainer

Method `setAbstract()`: Set abstract

Usage:

```
GSServiceSettings$setAbstract(abstract)
```

Arguments:

`abstract` abstract

Method `setAccessConstraints()`: Set access constraints

Usage:

```
GSServiceSettings$setAccessConstraints(accessConstraints)
```

Arguments:

`accessConstraints` access constraints

Method `setFees()`: Set fees

Usage:

```
GSServiceSettings$setFees(fees)
```

Arguments:

`fees` fees

Method `setKeywords()`: Set keywords

Usage:

```
GSServiceSettings$setKeywords(keywords)
```

Arguments:

`keywords` keywords

Method `addKeyword()`**:** Adds a keyword

Usage:

```
GSServiceSettings$addKeyword(keyword)
```

Arguments:

`keyword` keyword

Returns: `TRUE` if added, `FALSE` otherwise

Method `delKeyword()`**:** Deletes a keyword

Usage:

```
GSServiceSettings$delKeyword(keyword)
```

Arguments:

`keyword` keyword

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSServiceSettings$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
settings <- GSServiceSettings$new(service = "WMS")
settings$setEnabled(TRUE)
```

`GSShapefileDataStore` *Geoserver REST API ShapeFileDataStore*

Description

Geoserver REST API ShapeFileDataStore

Geoserver REST API ShapeFileDataStore

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer Shapefile dataStore

Methods

`new(xml, name, description, enabled, url)` Instantiates a GSShapefileDataStore object

`setUrl(url)` Set the spatial files data URL

`setCharset(charset)` Set the charset used for DBF file. Default value is 'ISO-8859-1'

`setCreateSpatialIndex(create)` Set the 'Create Spatial Index' option. Default is TRUE

`setMemoryMappedBuffer(buffer)` Set the 'Memory Mapped Buffer' option. Default is TRUE

`CacheReuseMemoryMaps(maps)` Set the 'Cache & Reuse Memory Maps' option. Default is TRUE

`setDefautConnectionParameters()` Set the defaut connection paramaters

Super classes

`geosapi::GSRESTResource` -> `geosapi::GSAbstractStore` -> `geosapi::GSAbstractDataStore` -> `GSShapefileDataStore`

Methods

Public methods:

- `GSShapefileDataStore$new()`
- `GSShapefileDataStore$setUrl()`
- `GSShapefileDataStore$setCharset()`
- `GSShapefileDataStore$setCreateSpatialIndex()`
- `GSShapefileDataStore$setMemoryMappedBuffer()`
- `GSShapefileDataStore$setCacheReuseMemoryMaps()`
- `GSShapefileDataStore$setDefautConnectionParameters()`
- `GSShapefileDataStore$clone()`

Method `new()`**:** initializes a shapefile data store

Usage:

```
GSShapefileDataStore$new(
  xml = NULL,
  name = NULL,
  description = "",
  enabled = TRUE,
  url
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`name` coverage store name

`description` coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

`url` url

Method `setUrl()`**:** Set the spatial files data URL

Usage:
```
GSShapefileDataStore$setUrl(url)
```
Arguments:
`url` url

Method `setCharset()`**:** Set the charset used for DBF file.
Usage:
```
GSShapefileDataStore$setCharset(charset = "ISO-8859-1")
```
Arguments:
`charset` charset. Default value is 'ISO-8859-1'

Method `setCreateSpatialIndex()`**:** Set the 'Create Spatial Index' option
Usage:
```
GSShapefileDataStore$setCreateSpatialIndex(create = TRUE)
```
Arguments:
`create` create. Default is `TRUE`

Method `setMemoryMappedBuffer()`**:** Set the 'Memory Mapped Buffer' option
Usage:
```
GSShapefileDataStore$setMemoryMappedBuffer(buffer = FALSE)
```
Arguments:
`buffer` buffer. Default is `FALSE`

Method `setCacheReuseMemoryMaps()`**:** Set the 'Cache & Reuse Memory Maps' option.
Usage:
```
GSShapefileDataStore$setCacheReuseMemoryMaps(maps = TRUE)
```
Arguments:
`maps` maps. Default is `TRUE`

Method `setDefautConnectionParameters()`**:** Set default connection parameters
Usage:
```
GSShapefileDataStore$setDefautConnectionParameters()
```

Method `clone()`**:** The objects of this class are cloneable with this method.
Usage:
```
GSShapefileDataStore$clone(deep = FALSE)
```
Arguments:
`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSShapefileDataStore$new(name = "ds", description = "des",
                         enabled = TRUE, url = "file://data/shape.shp")
```

GSShapefileDirectoryDataStore
Geoserver REST API ShapeFileDirectoryDataStore

Description

Geoserver REST API ShapeFileDirectoryDataStore

Geoserver REST API ShapeFileDirectoryDataStore

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer Shapefile directory dataStore

Super classes

`geosapi::GSRESTResource` -> `geosapi::GSAbstractStore` -> `geosapi::GSAbstractDataStore` -> `geosapi::GSShapefileDataStore` -> `GSShapefileDirectoryDataStore`

Methods

Public methods:

- `GSShapefileDirectoryDataStore$new()`
- `GSShapefileDirectoryDataStore$setUrl()`
- `GSShapefileDirectoryDataStore$setCharset()`
- `GSShapefileDirectoryDataStore$setCreateSpatialIndex()`
- `GSShapefileDirectoryDataStore$setMemoryMappedBuffer()`
- `GSShapefileDirectoryDataStore$setCacheReuseMemoryMaps()`
- `GSShapefileDirectoryDataStore$setDefautConnectionParameters()`
- `GSShapefileDirectoryDataStore$clone()`

Method `new()`**:** initializes a shapefile directory data store

Usage:

```
GSShapefileDirectoryDataStore$new(
  xml = NULL,
  name = NULL,
  description = "",
  enabled = TRUE,
  url
)
```

Arguments:

`xml` an object of class XMLInternalNode-class to create object from XML

`name` coverage store name
`description` coverage store description
`enabled` whether the store should be enabled or not. Default is `TRUE`
`url` url

Method `setUrl()`**:** Set the spatial files data URL

Usage:

```
GSShapefileDirectoryDataStore$setUrl(url)
```

Arguments:

`url` url

Method `setCharset()`**:** Set the charset used for DBF file.

Usage:

```
GSShapefileDirectoryDataStore$setCharset(charset = "ISO-8859-1")
```

Arguments:

`charset` charset. Default value is 'ISO-8859-1'

Method `setCreateSpatialIndex()`**:** Set the 'Create Spatial Index' option

Usage:

```
GSShapefileDirectoryDataStore$setCreateSpatialIndex(create = TRUE)
```

Arguments:

`create` create. Default is `TRUE`

Method `setMemoryMappedBuffer()`**:** Set the 'Memory Mapped Buffer' option

Usage:

```
GSShapefileDirectoryDataStore$setMemoryMappedBuffer(buffer = FALSE)
```

Arguments:

`buffer` buffer. Default is `FALSE`

Method `setCacheReuseMemoryMaps()`**:** Set the 'Cache & Reuse Memory Maps' option.

Usage:

```
GSShapefileDirectoryDataStore$setCacheReuseMemoryMaps(maps = TRUE)
```

Arguments:

`maps` maps. Default is `TRUE`

Method `setDefautConnectionParameters()`**:** Set default connection parameters

Usage:

```
GSShapefileDirectoryDataStore$setDefautConnectionParameters()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSShapefileDirectoryDataStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSShapefileDirectoryDataStore$new(name = "ds", description = "des",
                          enabled = TRUE, url = "file://data")
```

GSShinyMonitor *Geoserver REST API DataStore*

Description

Geoserver REST API DataStore

Geoserver REST API DataStore

Format

R6Class object.

Value

Object of R6Class for setting a GS Shiny monitoring app

Methods

Public methods:

- GSShinyMonitor$new()
- GSShinyMonitor$getMetric()
- GSShinyMonitor$run()
- GSShinyMonitor$clone()

Method `new()`**:** Initializes a Geoserver shiny monitoring tool

Usage:

```
GSShinyMonitor$new(manager, file = NULL, append = FALSE, sleep = 1)
```

Arguments:

`manager` object of class GSManager

`file` file File where to store monitoring results

`append` append. Whether results should be appended to existing file

`sleep` sleep. Interval in seconds to trigger monitor calls

Method `getMetric()`**:** Get metric

Usage:

```
GSShinyMonitor$getMetric(name)
```

Arguments:

`name` name

Returns: the Geoserver monitored metric

Method `run()`**:** Runs the application

Usage:

```
GSShinyMonitor$run()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSShinyMonitor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

Internal class used for `GSManager$monitor` method

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`GSStyleManager` *Geoserver REST API Style Manager*

Description

Geoserver REST API Style Manager

Geoserver REST API Style Manager

Format

`R6Class` object.

Value

Object of `R6Class` with methods for managing the styles of a GeoServer instance.

Methods

`new(url, user, pwd, logger)` This method is used to instantiate a GSManager with the `url` of the GeoServer and credentials to authenticate (`user`/`pwd`). By default, the `logger` argument will be set to `NULL` (no logger). This argument accepts two possible values: `INFO`: to print only geosapi logs, `DEBUG`: to print geosapi and CURL logs

`getStyles()`

`getStyleNames()`

`getStyle(style)`

`createStyle(file, sldBody, name, raw, ws)`

`updateStyle(file, sldBody, name, raw, ws)` Updates a GeoServer style. Returns `TRUE` if the style has been successfully updated, `FALSE` otherwise

`deleteStyle(style, recurse, purge, ws)` Deletes a GeoServer style given a name. Returns `TRUE` if the style has been successfully deleted, `FALSE` otherwise

`getSLDVersion(sldBody)` Get the SLD version from the XML object (of class `XMLInternalDocument`)

`getSLDBody(style, ws = NULL)` Get the SLD Body given a style name. This method is only supported for Geoserver >= 2.2.

Super class

`geosapi::GSManager` -> `GSStyleManager`

Methods

Public methods:

- `GSStyleManager$getStyles()`
- `GSStyleManager$getStyleNames()`
- `GSStyleManager$getStyle()`
- `GSStyleManager$createStyle()`
- `GSStyleManager$updateStyle()`
- `GSStyleManager$deleteStyle()`
- `GSStyleManager$getSLDVersion()`
- `GSStyleManager$getSLDBody()`
- `GSStyleManager$clone()`

Method `getStyles()`**:** Get the list of available styles.

Usage:

```
GSStyleManager$getStyles()
```

Returns: an object of class `list` containing items of class `GSStyle`

Method `getStyleNames()`**:** Get the list of available style names

Usage:

```
GSStyleManager$getStyleNames()
```

Returns: a vector of class `character`

Method `getStyle()`**:** Get a `GSStyle` object given a style name.

Usage:

```
GSStyleManager$getStyle(style, ws = NULL)
```

Arguments:

`style` style name

`ws` workspace name. Optional

Returns: object of class GSStyle

Method `createStyle()`**:** Creates a GeoServer style given a name.

Usage:

```
GSStyleManager$createStyle(file, sldBody = NULL, name, raw = FALSE, ws = NULL)
```

Arguments:

`file` file

`sldBody` SLD body

`name` name

`raw` raw

`ws` workspace name

Returns: `TRUE` if the style has been successfully created, `FALSE` otherwise

Method `updateStyle()`**:** Updates a GeoServer style given a name.

Usage:

```
GSStyleManager$updateStyle(file, sldBody = NULL, name, raw = FALSE, ws = NULL)
```

Arguments:

`file` file

`sldBody` SLD body

`name` name

`raw` raw

`ws` workspace name

Returns: `TRUE` if the style has been successfully updated, `FALSE` otherwise

Method `deleteStyle()`**:** Deletes a style given a name. By defaut, the option `recurse` is set to FALSE, ie datastore layers are not removed. To remove all coverage store layers, set this option to TRUE. The `purge` parameter is used to customize the delete of files on disk (in case the underlying reader implements a delete method).

Usage:

```
GSStyleManager$deleteStyle(name, recurse = FALSE, purge = FALSE, ws = NULL)
```

Arguments:

`name` name

`recurse` recurse

`purge` purge

`ws` workspace name

Returns: TRUE if the style has been successfully deleted, FALSE otherwise

Method `getSLDVersion()`**:** Get SLD version

Usage:

```
GSStyleManager$getSLDVersion(sldBody)
```

Arguments:

`sldBody` SLD body

Method `getSLDBody()`**:** Get SLD body

Usage:

```
GSStyleManager$getSLDBody(style, ws = NULL)
```

Arguments:

`style` style name

`ws` workspace name

Returns: an object of class XMLInternalNode-class

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSStyleManager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
## Not run:
   GSStyleManager$new("http://localhost:8080/geoserver", "admin", "geoserver")

## End(Not run)
```

`GSUtils` *Geoserver REST API Manager Utils*

Description

Geoserver REST API Manager Utils

Geoserver REST API Manager Utils

Format

`R6Class` object.

Value

Object of R6Class with static util methods for communication with the REST API of a GeoServer instance.

Static methods

`getUserAgent()` This method is used to get the user agent for performing GeoServer API requests. Here the user agent will be compound by geosapi package name and version.

`getUserToken(user, pwd)` This method is used to get the user authentication token for performing GeoServer API requests. Token is given a Base64 encoded string.

`GET(url, user, pwd, path, verbose)` This method performs a GET request for a given `path` to GeoServer REST API

`PUT(url, user, pwd, path, filename, contentType, verbose)` This method performs a PUT request for a given `path` to GeoServer REST API, to upload a file of name `filename` with given `contentType`

`POST(url, user, pwd, path, content, contentType, verbose)` This method performs a POST request for a given `path` to GeoServer REST API, to post content of given `contentType`

`DELETE(url, user, pwd, path, verbose)` This method performs a DELETE request for a given GeoServer resource identified by a `path` in GeoServer REST API

`parseResponseXML(req)` Convenience method to parse XML response from GeoServer REST API. Although package **httr** suggests the use of **xml2** package for handling XML, **geosapi** still relies on the package **XML**. Response from **httr** is retrieved as text, and then parsed as XML using `xmlParse` function.

`getPayloadXML(obj)` Convenience method to create payload XML to send to GeoServer.

`setBbox(minx, miny, maxx, maxy, bbox, crs)` Creates an list object representing a bbox. Either from coordinates or from a `bbox` object (matrix).

Methods

Public methods:

- `GSUtils$clone()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSUtils$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

GSVersion *A GeoServer version*

Description

This class allows to grab the GeoServer version. By default, a tentative is made to fetch version from web admin default page, since Geoserver REST API did not support GET operation for the Geoserver version in past releases of Geoserver.

Format

R6Class object.

Details

Geoserver REST API - Geoserver Version

Value

Object of R6Class for modelling a GeoServer version

Public fields

`version` version

`value` value

Methods

Public methods:

- `GSVersion$new()`
- `GSVersion$lowerThan()`
- `GSVersion$greaterThan()`
- `GSVersion$equalTo()`
- `GSVersion$clone()`

Method `new()`**:** Initializes an object of class GSVersion

Usage:

```
GSVersion$new(url, user, pwd)
```

Arguments:

`url` url

`user` user

`pwd` pwd

Method `lowerThan()`**:** Compares to a version and returns TRUE if it is lower, FALSE otherwise

Usage:

```
GSVersion$lowerThan(version)
```

Arguments:

`version` version

Returns: `TRUE` if lower, `FALSE` otherwise

Method `greaterThan()`**:** Compares to a version and returns TRUE if it is greater, FALSE otherwise

Usage:

```
GSVersion$greaterThan(version)
```

Arguments:

`version` version

Returns: `TRUE` if greater, `FALSE` otherwise

Method `equalTo()`**:** Compares to a version and returns TRUE if it is equal, FALSE otherwise

Usage:

```
GSVersion$equalTo(version)
```

Arguments:

`version` version

Returns: `TRUE` if equal, `FALSE` otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSVersion$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
## Not run:
version <- GSVersion$new(
             url = "http://localhost:8080/geoserver",
             user = "admin", pwd = "geoserver"
           )

## End(Not run)
```

GSVirtualTable *Geoserver REST API GSVirtualTable*

Description

Geoserver REST API GSVirtualTable

Geoserver REST API GSVirtualTable

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer virtual table

Methods

`new(xml)` This method is used to instantiate a GSVirtualTable

`decode(xml)` This method is used to decode a GSVirtualTable from XML

`encode()` This method is used to encode a GSVirtualTable to XML

`setName(name)` Sets the name of the virtual table

`setSql(sql)` Sets the sql of the virtual table

`setEscapeSql(escapeSql)` Sets the escapeSql. Default is FALSE

`setKeyColumn(keyColumn)` Sets the keyColumn. Name of the column to be the primary key

`setGeometry(vtg)` Sets the virtual table geometry

`addParameter(parameter)` Adds a virtual table parameter

`delParameter(parameter)` Removes a virtual table parameter.

Super class

`geosapi::GSRESTResource` -> `GSVirtualTable`

Public fields

`name` name

`sql` SQL statement

`escapeSql` escape SQL?

`keyColumn` key column

`geometry` geometry

`parameters` list of virtual parameters

Methods

Public methods:

- GSVirtualTable$new()
- GSVirtualTable$decode()
- GSVirtualTable$setName()
- GSVirtualTable$setSql()
- GSVirtualTable$setEscapeSql()
- GSVirtualTable$setKeyColumn()
- GSVirtualTable$setGeometry()
- GSVirtualTable$addParameter()
- GSVirtualTable$delParameter()
- GSVirtualTable$clone()

Method `new()`**:** Initializes an object of class GSVirtualTable

Usage:

```
GSVirtualTable$new(xml = NULL)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `decode()`**:** Decodes from XML

Usage:

```
GSVirtualTable$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setName()`**:** Set name

Usage:

```
GSVirtualTable$setName(name)
```

Arguments:

`name` name

Method `setSql()`**:** Set SQL

Usage:

```
GSVirtualTable$setSql(sql)
```

Arguments:

`sql` sql

Method `setEscapeSql()`**:** Set escape SQL

Usage:

```
GSVirtualTable$setEscapeSql(escapeSql)
```

Arguments:

`escapeSql` escape SQL

Method `setKeyColumn()`: Set key column

Usage:

```
GSVirtualTable$setKeyColumn(keyColumn)
```

Arguments:

`keyColumn` key column

Method `setGeometry()`: Set geometry

Usage:

```
GSVirtualTable$setGeometry(vtg)
```

Arguments:

`vtg` object of class GSVirtualTableGeometry

Method `addParameter()`: Adds parameter

Usage:

```
GSVirtualTable$addParameter(parameter)
```

Arguments:

`parameter` object of class GSVirtualTableParameter

Returns: `TRUE` if added, `FALSE` otherwise

Method `delParameter()`: Deletes parameter

Usage:

```
GSVirtualTable$delParameter(parameter)
```

Arguments:

`parameter` object of class GSVirtualTableParameter

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
GSVirtualTable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSVirtualTable$new()
```

GSVirtualTableGeometry

Geoserver REST API GSVirtualTableGeometry

Description

Geoserver REST API GSVirtualTableGeometry

Geoserver REST API GSVirtualTableGeometry

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer virtual table geometry

Super class

geosapi::GSRESTResource -> GSVirtualTableGeometry

Public fields

name geometry name

type geometry type

srid geometry SRID

Methods

Public methods:

- GSVirtualTableGeometry$new()
- GSVirtualTableGeometry$decode()
- GSVirtualTableGeometry$clone()

Method new(): Initializes an object of class GSVirtualTableGeometry

Usage:

```
GSVirtualTableGeometry$new(xml = NULL, name, type, srid)
```

Arguments:

xml object of class XMLInternalNode-class

name name

type type

srid srid

Method decode(): Decodes from XML

Usage:

```
GSVirtualTableGeometry$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSVirtualTableGeometry$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSVirtualTableGeometry$new(name = "work", type = "MultiPolygon", srid = 4326)
```

`GSVirtualTableParameter`

Geoserver REST API GSVirtualTableParameter

Description

Geoserver REST API GSVirtualTableParameter

Geoserver REST API GSVirtualTableParameter

Format

`R6Class` object.

Value

Object of `R6Class` for modelling a GeoServer virtual table parameter

Super class

`geosapi::GSRESTResource` -> `GSVirtualTableParameter`

Public fields

`name` parameter name

`defaultValue` parameter default value

`regexpValidator` parameter regexp validator

Methods

Public methods:

- GSVirtualTableParameter$new()
- GSVirtualTableParameter$decode()
- GSVirtualTableParameter$clone()

Method `new()`**:** Initializes an object of class GSVirtualTableParameter

Usage:

```
GSVirtualTableParameter$new(xml = NULL, name, defaultValue, regexpValidator)
```

Arguments:

`xml` object of class XMLInternalNode-class

`name` name

`defaultValue` default value

`regexpValidator` regexp validator

Method `decode()`**:** Decodes from XML

Usage:

```
GSVirtualTableParameter$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSVirtualTableParameter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSVirtualTableParameter$new(name = "fieldname", defaultValue = "default_value",
                            regexpValidator = "someregexp")
```

GSWorkspace *Geoserver REST API Workspace*

Description

Geoserver REST API Workspace

Geoserver REST API Workspace

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer workspace

Super class

geosapi::GSRESTResource -> GSWorkspace

Public fields

name name

Methods

Public methods:

- GSWorkspace$new()
- GSWorkspace$decode()
- GSWorkspace$clone()

Method new(): initializes a GSWorkspace

Usage:

```
GSWorkspace$new(xml = NULL, name)
```

Arguments:

xml an object of class XMLInternalNode-class

name name

Method decode(): Decodes from XML

Usage:

```
GSWorkspace$decode(xml)
```

Arguments:

xml an object of class XMLInternalNode-class

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
GSWorkspace$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
GSWorkspace$new(name = "work")
```

GSWorkspaceManager *Geoserver REST API Workspace Manager*

Description

Geoserver REST API Workspace Manager

Geoserver REST API Workspace Manager

Format

R6Class object.

Value

Object of R6Class with methods for managing the workspaces of a GeoServer instance.

Super class

geosapi::GSManager -> GSWorkspaceManager

Methods

Public methods:

- GSWorkspaceManager$getWorkspaces()
- GSWorkspaceManager$getWorkspaceNames()
- GSWorkspaceManager$getWorkspace()
- GSWorkspaceManager$createWorkspace()
- GSWorkspaceManager$updateWorkspace()
- GSWorkspaceManager$deleteWorkspace()
- GSWorkspaceManager$getWorkspaceSettings()
- GSWorkspaceManager$createWorkspaceSettings()
- GSWorkspaceManager$updateWorkspaceSettings()

- GSWorkspaceManager$deleteWorkspaceSettings()
- GSWorkspaceManager$clone()

Method `getWorkspaces()`**:** Get the list of available workspace. Returns an object of class `list` containing items of class GSWorkspace

Usage:

```
GSWorkspaceManager$getWorkspaces()
```

Arguments:

`a` list of GSWorkspace

Method `getWorkspaceNames()`**:** Get the list of available workspace names. Returns an vector of class `character`

Usage:

```
GSWorkspaceManager$getWorkspaceNames()
```

Returns: a list of workspace names

Method `getWorkspace()`**:** Get a GSWorkspace object given a workspace name.

Usage:

```
GSWorkspaceManager$getWorkspace(ws)
```

Arguments:

`ws` workspace name

Returns: an object of class GSWorkspace

Method `createWorkspace()`**:** Creates a GeoServer workspace given a name, and an optional URI. If the URI is not specified, GeoServer will automatically create an associated Namespace with the URI being "http://workspaceName. If the URI is specified, the method invokes the method `createNamespace(ns, uri)` of the GSNamespaceManager. Returns `TRUE` if the workspace has been successfully created, `FALSE` otherwise

Usage:

```
GSWorkspaceManager$createWorkspace(name, uri)
```

Arguments:

`name` name

`uri` uri

Returns: `TRUE` if created, `FALSE` otherwise

Method `updateWorkspace()`**:** Updates a GeoServer workspace given a name, and an optional URI. If the URI is not specified, GeoServer will automatically update the associated Namespace with the URI being "http://workspaceName. If the URI is specified, the method invokes the method `updateNamespace(ns, uri)` of the GSNamespaceManager. Returns `TRUE` if the workspace has been successfully updated, `FALSE` otherwise

Usage:

```
GSWorkspaceManager$updateWorkspace(name, uri)
```

Arguments:

name name
uri uri
Returns: TRUE if created, FALSE otherwise

Method `deleteWorkspace()`**:** Deletes a GeoServer workspace given a name.
Usage:
```
GSWorkspaceManager$deleteWorkspace(name, recurse = FALSE)
```
Arguments:
name name
recurse recurse
Returns: TRUE if the workspace has been successfully deleted, FALSE otherwise

Method `getWorkspaceSettings()`**:** Updates workspace settings
Usage:
```
GSWorkspaceManager$getWorkspaceSettings(ws)
```
Arguments:
ws workspace name
Returns: an object of class GSWorkspaceSettings

Method `createWorkspaceSettings()`**:** Creates workspace settings
Usage:
```
GSWorkspaceManager$createWorkspaceSettings(ws, workspaceSettings)
```
Arguments:
ws workspace name
workspaceSettings object of class GSWorkspaceSettings
Returns: TRUE if created, FALSE otherwise

Method `updateWorkspaceSettings()`**:** Updates workspace settings
Usage:
```
GSWorkspaceManager$updateWorkspaceSettings(ws, workspaceSettings)
```
Arguments:
ws workspace name
workspaceSettings object of class GSWorkspaceSettings
Returns: TRUE if updated, FALSE otherwise

Method `deleteWorkspaceSettings()`**:** Deletes workspace settings
Usage:
```
GSWorkspaceManager$deleteWorkspaceSettings(ws)
```
Arguments:
ws workspace name
Returns: TRUE if deleted, FALSE otherwise

Method `clone()`**:** The objects of this class are cloneable with this method.
Usage:
```
GSWorkspaceManager$clone(deep = FALSE)
```
Arguments:
deep Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
## Not run:
   GSWorkspaceManager$new("http://localhost:8080/geoserver", "admin", "geoserver")

## End(Not run)
```

GSWorkspaceSettings *Geoserver REST API Workspace Setting*

Description

Geoserver REST API Workspace Setting

Geoserver REST API Workspace Setting

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer workspace settings

Super class

geosapi::GSRESTResource -> GSWorkspaceSettings

Public fields

`contact` contact

`charset` charset

`numDecimals` number of decimal

`onlineResource` online resource

`verbose` verbose

`verboseExceptions` verbose exceptions

`localWorkspaceIncludesPrefix` local workspace includes prefix

Methods

Public methods:

- GSWorkspaceSettings$new()
- GSWorkspaceSettings$decode()
- GSWorkspaceSettings$setCharset()
- GSWorkspaceSettings$setNumDecimals()
- GSWorkspaceSettings$setOnlineResource()
- GSWorkspaceSettings$setVerbose()
- GSWorkspaceSettings$setVerboseExceptions()
- GSWorkspaceSettings$setLocalWorkspaceIncludesPrefix()
- GSWorkspaceSettings$clone()

Method `new()`**:** This method is used to instantiate a `GSWorkspaceSettings`. This settings object is required to activate a workspace configuration, using the method `GSManager$createWorkspaceSettings`. Supported from GeoServer 2.12

Usage:

```
GSWorkspaceSettings$new(xml = NULL)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `decode()`**:** Decodes from XML

Usage:

```
GSWorkspaceSettings$decode(xml)
```

Arguments:

`xml` object of class XMLInternalNode-class

Method `setCharset()`**:** Set charset

Usage:

```
GSWorkspaceSettings$setCharset(charset)
```

Arguments:

`charset` charset

Method `setNumDecimals()`**:** Set number of decimals

Usage:

```
GSWorkspaceSettings$setNumDecimals(numDecimals)
```

Arguments:

`numDecimals` number of decimals

Method `setOnlineResource()`**:** Set online resource

Usage:

```
GSWorkspaceSettings$setOnlineResource(onlineResource)
```

Arguments:

`onlineResource` online resource

Method `setVerbose()`**:** Set verbose

Usage:

```
GSWorkspaceSettings$setVerbose(verbose)
```

Arguments:

`verbose` verbose

Method `setVerboseExceptions()`**:** Set verbose exceptions

Usage:

```
GSWorkspaceSettings$setVerboseExceptions(verboseExceptions)
```

Arguments:

`verboseExceptions` verbose exceptions

Method `setLocalWorkspaceIncludesPrefix()`**:** Set local workspace includes prefix

Usage:

```
GSWorkspaceSettings$setLocalWorkspaceIncludesPrefix(includesPrefix)
```

Arguments:

`includesPrefix` includes prefix

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSWorkspaceSettings$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
settings <- GSWorkspaceSettings$new()
settings$setCharset("UTF-8")
settings$setNumDecimals(5)
```

GSWorldImageCoverageStore

Geoserver REST API WorldImageCoverageStore

Description

Geoserver REST API WorldImageCoverageStore

Geoserver REST API WorldImageCoverageStore

Format

R6Class object.

Value

Object of R6Class for modelling a GeoServer WorldImage CoverageStore

Super classes

geosapi::GSRESTResource -> geosapi::GSAbstractStore -> geosapi::GSAbstractCoverageStore -> GSWorldImageCoverageStore

Public fields

url url

Methods

Public methods:

- GSWorldImageCoverageStore$new()
- GSWorldImageCoverageStore$clone()

Method new(): Initializes an WorldImage coverage store

Usage:

```
GSWorldImageCoverageStore$new(
  xml = NULL,
  name = NULL,
  description = "",
  enabled = TRUE,
  url = NULL
)
```

Arguments:

xml an object of class XMLInternalNode-class to create object from XML

name coverage store name

description coverage store description

`enabled` whether the store should be enabled or not. Default is `TRUE`

`url` url

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GSWorldImageCoverageStore$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Index