Importazione ed Esportazione Dati API JS

Geoweb espone api js per attuare l'importazione e l'esportazione di dati con formati standard.

Formati supportati:

- **.xlsx**
- **.csv**
- **.shp** (per dati spaziali)

Prerequisiti

Tabella dell'entità da gestire presente nel DB, con configurazione della relativa classe, e di tutti gli widget necessari.

Implementazione delle opportune azioni javascript di import/export.

Le azioni si differenziano a seconda del tipo (CSV, XLSX o SHP), ma hanno anche in comune alcuni aspetti e modalità di utilizzo.

Per le procedure di import presenza della tabella descritta qui.

CSV

EXPORT CSV (js API)

Esistono attualmente 2 casi d'uso per l'export csv:

1. **CASO 1: Export per aggiornamento**, con **intestazioni Codici** (columnName degli attributi). Usato al fine di essere successivamente reimportato (nell'ambito dei casi d'uso di popolamento massivo, aggiornamento puntuale, riconciliazione dati, etc..)
2. **CASO 2: Export per integrazione con sistemi terzi**, con **intestazioni Personalizzate** (come da remapping). Destinato a produrre file secondo il formato di istemi esterni, per integrazioni di vario genere.

exportCSV()

(from 4.7.4)

CASO 1: Export per aggiornamento CASO 2: Export per integrazione con sistemi terzi

Funzione generica. Di base lavora con gli attributi in lista per la classe. Gli header delle colonne esportate possono essere rimappati.

Parametri

- **gwClassName**: String, required
- **queryParameter**: Object, optional
- **options**: Object, optional
    - **columnHeadersMap**: Object, optional, Map<String,String>

Esempi

```
var gwClassName = 'gwClassName';
var queryParameter = {filters: [..]};
exportCSV(gwClassName, queryParameter);
```

```
var gwClassName = 'gwClassName';
var queryParameter = {filters: [..]};
var columnHeadersMap = {'gw_attr_name': 'target_attribute_name'}
var options = {columnHeadersMap: columnHeadersMap};
exportCSV(gwClassName, queryParameter, options);
```

exportCSVListSelected()

(from 4.7.4)

CASO 1: Export per aggiornamento CASO 2: Export per integrazione con sistemi terzi

Funzione specifica per la gwClassList. Lavora con gli attributi in lista per la classe. Gli header delle colonne esportate possono essere rimappati.

Parametri

- **grid**: Object, required
- **options**: Object, optional
    - **columnHeadersMap**: Object, optional, Map<String,String>

Esempi

```
exportCSVListSelected(grid);
```

```
var columnHeadersMap = {'gw_attr_name': 'target_attribute_name'};
var options = {columnHeadersMap: columnHeadersMap};
exportCSVListSelected(grid, options);
```

exportCSVAllAttributes()

(from 4.7.4)

CASO 1: Export per aggiornamento CASO 2: Export per integrazione con sistemi terzi

Funzione generica. Lavora con tutti gli attributi della classe (collegati ad almeno un gruppo attributo).

Gli header delle colonne esportate possono essere rimappati.

Parametri

- **gwClassName**: String, required
- **queryParameter**: Object, optional
- **options**: Object, optional
    - **columnHeadersMap**: Object, optional, Map<String,String>

Esempi

```
var gwClassName = 'gwClassName';
var queryParameter = {filters: [..]};
exportCSVAllAttributes(gwClassName, queryParameter);
```

```
var queryParameter = {filters: [..]};
var columnHeadersMap = {'gw_attr_name': 'target_attribute_name'};
var options = {columnHeadersMap: columnHeadersMap};
exportCSVAllAttributes(gwClassName, queryParameter, options);
```

exportCSVListSelectedAllAttributes()

(from 4.7.4)

CASO 1: Export per aggiornamento CASO 2: Export per integrazione con sistemi terzi

Funzione specifica per la gwClassList. Lavora con tutti gli attributi della classe (collegati ad almeno un gruppo attributo). Gli header delle colonne esportate possono essere rimappati.

Parametri

- **grid**: Object, required
- **options**: Object, optional
    - **columnHeadersMap**: Object, optional, Map<String,String>

Esempi

```
exportCSVListSelectedAllAttributes(grid);
```

```
var columnHeadersMap = {'gw_attr_name': 'target_attribute_name'};
var options = {columnHeadersMap: columnHeadersMap};
exportCSVListSelectedAllAttributes(grid, options);
```

exportCSVSpecifiedAttributes()

(from 4.7.4)

CASO 1: Export per aggiornamento CASO 2: Export per integrazione con sistemi terzi

Funzione generica. Lavora con gli attributi specificati come parametro. Gli header delle colonne esportate possono essere rimappati.

Parametri

- **grid**: Object, required
- **options**: Object, optional
  - **columnHeadersMap**: Object, optional, Map<String,String>
  - **specifiedAttributesNameList**: String[], required

Esempi

```
var gwClassName = 'gwClassName';
var queryParameter = {filters: [..]};
var columnHeadersMap = {'gw_attr_name': 'target_attribute_name'};
var specifiedAttributesNameList = ['attr_name_1', 'attr_name_2',
'attr_name_3'];
var options = {columnHeadersMap: columnHeadersMap,
specifiedAttributesNameList: specifiedAttributesNameList};
exportCSVSpecifiedAttributes(gwClassName, queryParameter, options);
```

exportCSVListSelectedSpecifiedAttributes()

(from 4.7.4)

CASO 1: Export per aggiornamento CASO 2: Export per integrazione con sistemi terzi

Funzione specifica per la gwClassList. Lavora con gli attributi specificati come parametro. Gli header delle colonne esportate possono essere rimappati.

Parametri

- **grid**: Object, required
- **options**: Object, optional
  - **columnHeadersMap**: Object, optional, Map<String,String>
  - **specifiedAttributesNameList**: String[], required

Esempi

```
var columnHeadersMap = {'gw_attr_name': 'target_attribute_name'};
var specifiedAttributesNameList = ['attr_name_1', 'attr_name_2',
'attr_name_3'];
var options = {columnHeadersMap: columnHeadersMap,
specifiedAttributesNameList: specifiedAttributesNameList};
exportCSVListSelectedSpecifiedAttributes(grid, options);
```

IMPORT CSV (js API)

importCSV()

Parametri

- **grid**: Object, required
- **options**: Object, required, contiene i seguenti parametri:

tipologia	Nome	Required	Descrizione breve
Integer	importType	true	è un numero intero ed indica il tipo di importazione. Può assumere il valore 0 o 1 o 2 o 3 o 4 come da sezione dedicata
function	callback	false	permette di eseguire la funzione di callback dopo l'importazione
String	childClassName	false	nome della classe geoweb referenziata da un widget **childList**. Da usare quando si vogliono importare anche i dati nella tabella della classe figlia. I record sono caricati già collegati al record padre dal quale viene lanciata l'azione
String	itemId	false	chiave del record da cui si sta lanciando l'azione, data.itemDB.pk_column, Da usare quando si vogliono importare anche i dati nella tabella della classe figlia. I record sono caricati già collegati al record padre dal quale viene lanciata l'azione
String	relationName	false	nome della gwRelation di classe, qualora si vuole importare i dati in una tabelle di una classe collegata a questa tramite widget childlist. I record sono caricati già collegati al record padre dal quale viene lanciata l'azione
String	codColumn	false	singola colonna o colonne multiple separate da virgola ','. Nome della o delle colonne per le quali, qualora si facesse un importazione tipo update, verrebbero usate come criterio di sovrascrittura. Le colonne se più di una devono essere separate da virgola
Object	columnHeadersMap	false	HashMap per la mappatura degli header del file con quelle del database. Vanno inserite soltanto quelle dove il nome è diverso e le si voglia importare. La forma è la seguente, nameColumnCsv: 'nameColumnDatabase'
Object	additionalMap	false	HashMap contenente la lista delle colonne non presenti nel file che si voglia comunque popolare durante l inserimento. La forma è nameColumnDatabase: 'valore', il valore può essere anche preso dinamicamente dalla sessione
String	scriptName	false	Nome dello scripts groovy, CON l'estensione che verrà lanciato al termine del caricamento. Lo script deve essere presente nella cartella dei contenuti statici sottocartella groovy

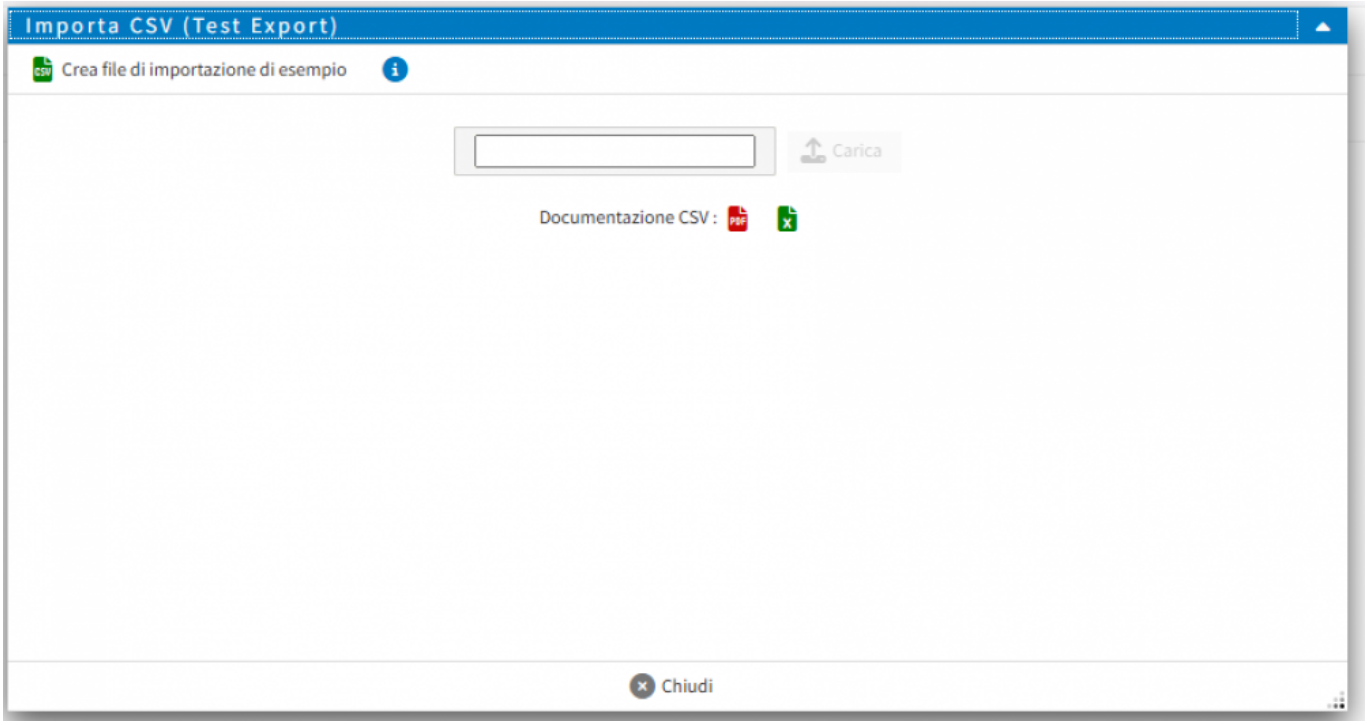
tipologia	Nome	Required	Descrizione breve
Integer	handleErrorsType	false	Valori ammessi: [0,1]. Deafault 0. Specifica il comportamento del sistema in presenza di un errore. Se 0 (default) il sistema va avanti segnalando alla fine i record non importati ed il relativo verificatosi. Se posto a 1 blocca l'intera importazione se almeno un record non può essere importato
Integer	idProcessImport	false	Se indicato non viene inserito un nuovo record nella tabella contenente l'archivio storico delle importazioni, bensì viene aggiornato il record con quello specifico idProcess
Integer	itemNotFoundAction	false	Valori ammessi: [0,1]. Default: 0. Valutato solo quando importType vale 2 (update). Quando a 1 fa si che tutti i record vengano aggiornati, impostando la property con nome uguale a itemNotFoundUpdateField con il valore itemNotFoundUpdateValue
String	itemNotFoundUpdateField	false	String, default null. Chiave usata quando itemNotFoundAction==1 && importType==2
String	itemNotFoundUpdateValue	false	String, default null. Valore usato quando itemNotFoundAction==1 && importType==2
String	csvTemplateLabel	false	(from 4.4.16) personalizza l'etichetta del pulsante del download del template
Boolean	hideDocumentationButtons	false	(from 4.4.16) Valori ammessi: [true, false]. Default false. Quando a true nasconde la documentazione e i relativi pulsanti che consentono, rispettivamente, il download del pdf e del file .xlsx

Esempi

Apre un dialog che permette di selezionare il file .csv da importare.

```
var importType = 0 ; //allowed 0, 1, 2, 3, 4
importCSV(grid, { importType: importType });
```

Dal dialog è possibile scaricare un file .csv di esempio di importazione avente, per default, come label la stringa *Crea file di importazione di esempio*.



(form 4.4.16) E' possibile personalizzare l'etichetta del pulsante, passando tra i parametri opzionali dell'azione importCSV il parametro **csvTemplateLabel** come nell'esempio:

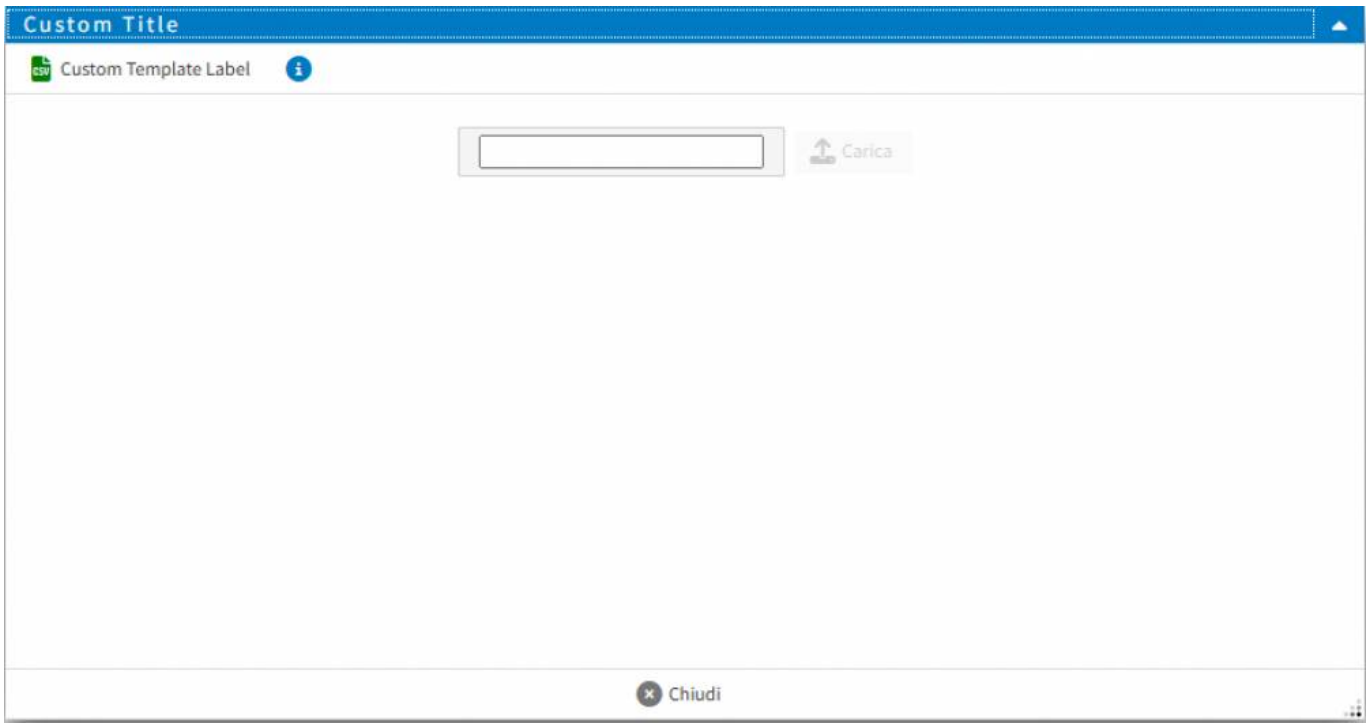
```
var importType = 0 ; //allowed 0, 1, 2, 3, 4
importCSV(grid,{importType: importType, csvTemplateLabel:'ESEMPIO LABEL CUSTOM'});
```

Oltre all'etichetta del pulsante che consente il download del template di tipo .csv, è possibile decidere se nascondere la documentazione .csv e i relativi pulsanti che consentono, rispettivamente, il download del pdf e del file xlsx passando tra i parametri opzionali dell'azione importCSV il parametro **hideDocumentationButtons** valorizzato a true come nell'esempio:

```
var importType = 0 ; //allowed 0, 1, 2, 3, 4
importCSV(grid,{importType: importType, hideDocumentationButtons: true});
```

Nell'immagine sottostante si riporta la maschera di importazione csv ottenuta ponendo hideDocumentationButtons uguale a true e csvTemplateLabel valorizzata come nell'esempio

```
var importType = 0 ; //allowed 0, 1, 2, 3, 4
importCSV(grid,{importType: importType, hideDocumentationButtons: true,
csvTemplateLabel: 'ESEMPIO LABEL CUSTOM'});
```



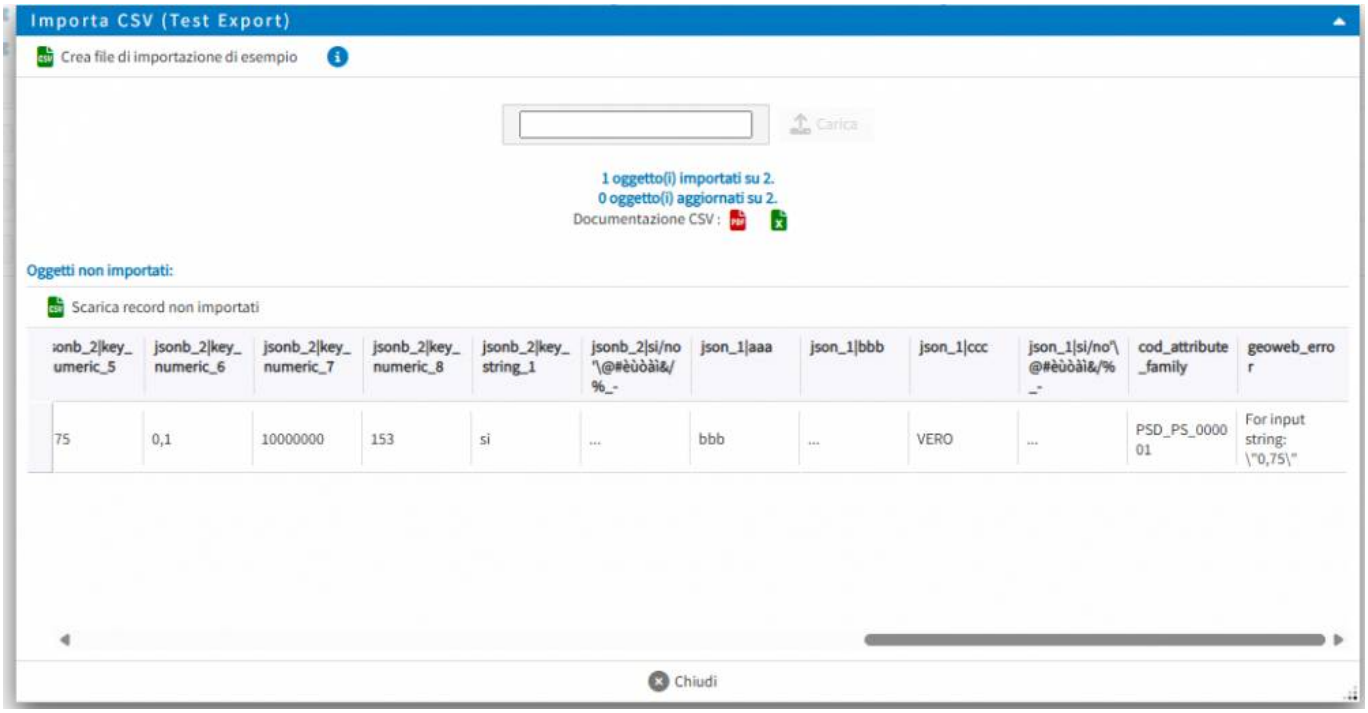
Importazione per Aggiornamento:

```
importCSV(grid, {importType: 2, codColumn: 'cod_item_mc,tenant_code'});
```

Importazione accodamento dei record di una classe figlia dal dettaglio di un record della classe padre, legati da una childlist:

```
importCSV(
    {gwClassName: 'CLASSE_PADRE'},
    {
        importType: 0,
        codColumn: 'NOME_CAMPO_CODICE',
        childClassName: 'NOME_CLASSE_FIGLIA',
        itemId: data.itemDB.CHIAVE_CLASSE_PADRE,
        relationName: 'NOME_RELAZIONE_CLASSE_PADRE_FIGLIA'
    }
);
```

In caso di errori, in base alla configurazione, il flusso di importazione può continuare senza bloccarsi, presentando poi alla fine un report sui record importati e su quelli non importati (con l'errore relativo). E' anche possibile scaricare un resoconto sui record non importati.



XLSX

EXPORT XLSX

L'esportazione in formato .xlsx è implementabile nelle seguenti modalità:

- definizione di una specifcia Report (jasper) in formato .xlsx e relative modalità di fruizione. Si veda a tal proposito(Creazione di Report)
- definizione di una azione di export. Di seguito varie function di api utilizzabili.

Esistono attualmente 3 casi d'uso per l'export xlsx:

1. **CASO 1: Export per consultazione**, con **intestazioni Etichette** (label degli attributi). Usato, come strumento di reportistica o comunque per eseguire elaborazioni di vario tipo in Excel, e comunque sempre senza necessità di essere reimportato in seguito
2. **CASO 2: Export per aggiornamento**, con **intestazioni Codici** (columnName degli attributi). Usato al fine di essere successivamente reimportato (nell'ambito dei casi d'uso di popolamento massivo, aggiornamento puntuale, riconciliazione dati, etc..)
3. **CASO 3: Export per integrazione con sistemi terzi**, con **intestazioni Personalizzate** (come da remapping). Destinato a produrre file secondo il formato di istemi esterni, per integrazioni di vario genere.

In base al caso d'uso da implementare va scelta la corretta api js.

standardListSelectedAllAttributeExportToExcel()

CASO 1: Export per consultazione

Esportazione Excel di **TUTTI** gli attributi della classe. Le intestazioni delle colonne sono le **Etichette**

definite negli attributi.

```
standardListSelectedAllAttributeExportToExcel(queryParameter, grid);
```

standardListSelectedExportToExcel()

CASO 1: Export per consultazione

Vengono esportati gli attributi della classe che sono **visibili in Lista**. Le intestazioni delle colonne sono le **Etichette** definite negli attributi.

```
standardListSelectedExportToExcel(queryParameter, grid);
```

standardListSelectedGroupAttributeExportToExcel()

CASO 1: Export per consultazione

(from 4.4.4) Vengono esportati **solo gli attributi di un gruppo attributo** della classe. Le intestazioni delle colonne sono le **Etichette** definite negli attributi.

```
standardListSelectedGroupAttributeExportToExcel(queryParameter, grid,
groupName);
```

standardListSelectedAllAttributeExportToSimpleExcel()

CASO 2: Export per aggiornamento

esporta un file CSV da modificare a mano e reimportare successivamente

Vengono esportati tutti gli attributi della classe.

Le intestazioni delle colonne sono i **Nomi** degli attributi.

Questa modalità di esportazione produce un file conforme al Template da utilizzare per l'importazione.

Il file va in ogni caso salvato in formato .csv.

Fare attenzione ai formati delle colonne, Excel potrebbe 'interpretare' i dati introducendo degli errori.

```
standardListSelectedAllAttributeExportToSimpleExcel(queryParameter, grid);
```

standardListSelectedGroupAttributeExportToSimpleExcel()

CASO 2: Export per aggiornamento

(from 4.4.4) Vengono esportati **solo gli attributi di un gruppo attributo** della classe. Le intestazioni delle colonne sono i **Nomi** degli attributi.

```
standardListSelectedGroupAttributeExportToSimpleExcel(queryParameter, grid, groupName);
```

standardPrefilterdListSelectedExportToExcel()

CASO 1: Export per consultazione

@Deprecated (usare standardListSelectedAllAttributeExportToExcel()) Vengono esportati **tutti gli attributi** della classe. Le intestazioni delle colonne sono le **Etichette** definite negli attributi.

```
standardPrefilterdListSelectedExportToExcel(queryParameter, grid);
```

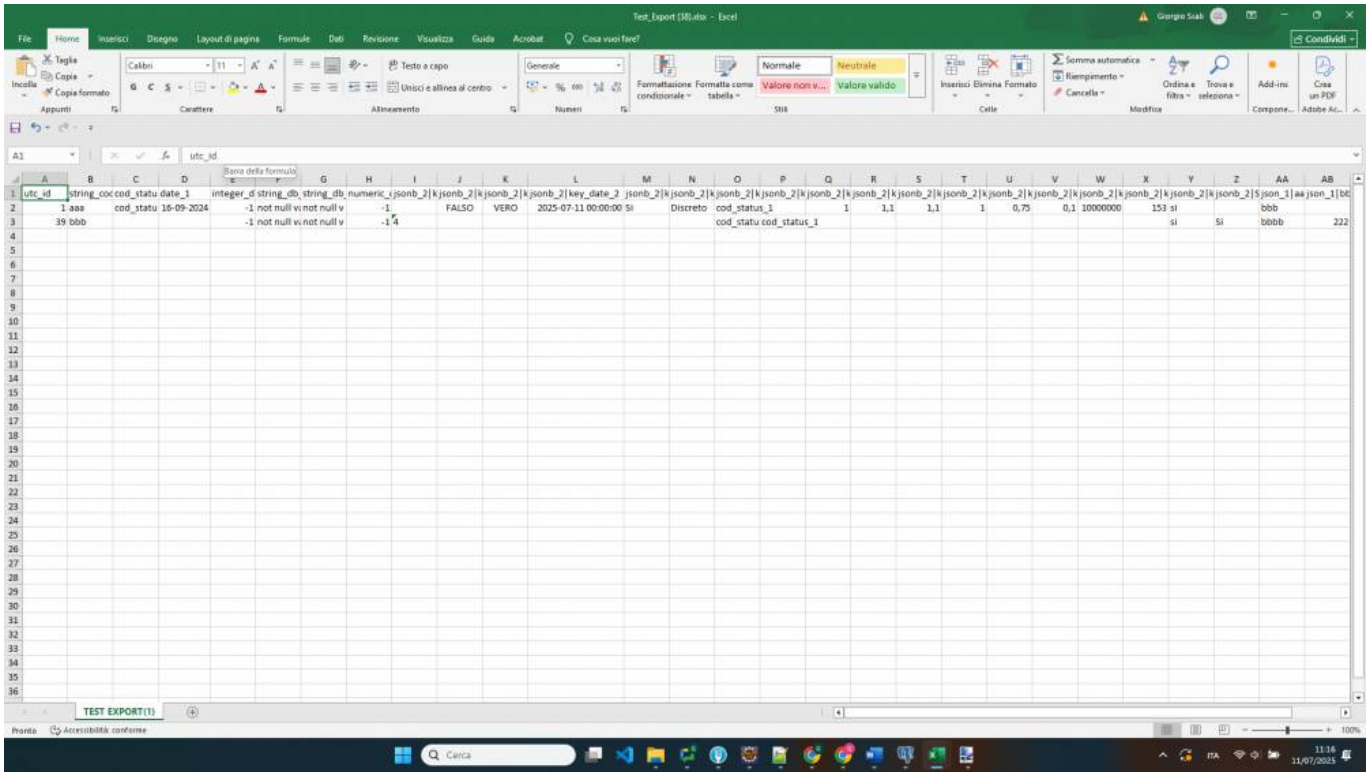
IMPORT XLSX

L'import dei file.xlsx viene ricondotto al flusso dell'import dei file .csv. Queste sono le fasi:

1. export di un file .xlsx.
2. modifica dei dati del file
3. conversione da .xlsx a .csv
4. utilizzo delle funzioni di import csv

Fase 1 - export di un file .xlsx

Sono importabili ESCLUSIVAMENTE i file ad uso aggiornamento Il file si presenta con la prima riga delle intestazioni di colonna, contenente **codici** (derivanti dai columnName degli attributi o da rimappaggio). Nel caso siano presenti ulteriori intestazioni di titolo o label negli header il file prodotto non destinato ad essere importato.



Fase 2 - modifica dei dati del file

Utilizzare Excel od equivalenti per modificare il dato. Vanno mantenuti eventuali formati del dato, come già presenti nei dati di esportazione

- i valori data vanno espressi nei formati

  ```
  dd-MM-yyyy HH:mm:ss
  ```

  o

  ```
  dd-MM-yyyy
  ```

  . Esempi:

  ```
  11-21-2025  00:00:00
  ```

  o

  ```
  11-21-2025
  ```

- i valori data dentro un JSON (si riconoscono in quanto nella cella di intestazione è sempre presente il char separatore |) vanno espressi nel formato

  ```
  dd-MM-yyyy HH:mm:ss
  ```

  . Esempio:

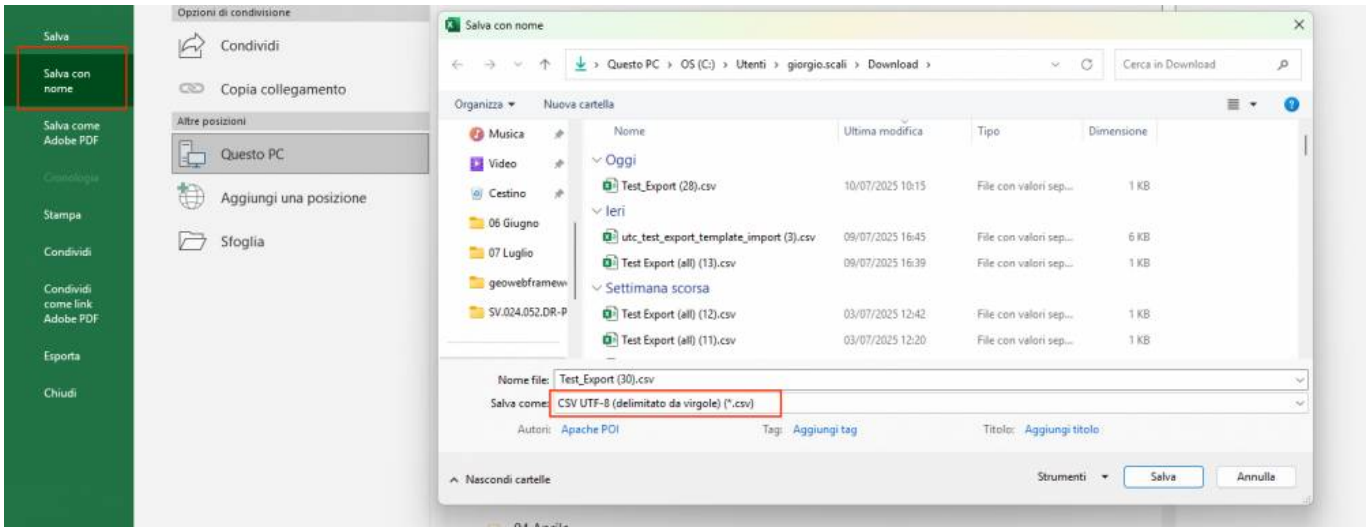
```
11-21-2025 00:00:00
```

- Per i tipi di dato decimale, rispettare l'uso della virgola o del punto come separatore in base alle impostazioni del sistema locale.

Fase 3 - conversione da .xlsx a .csv

Il file va salvato in formato .csv. Operazioni da eseguire in MS Office Excel:

- File ⇒ Salva con nome
- sceglie la posizione nel filesystem
- Selezionare il formato **“CSV UTF-8 (delimitato da virgole) (*.csv)“**



- Salva
- verificare, aprendo il file con un qualsiasi editor di testo (come Notepad), che le intestazioni ed i dati siano delimitati da **;** (*punto e virgola*) e non da semplice **,** (*virgola*)

Se fossero presenti delle **,** al posto dei **;**, ciò dipende dalle impostazioni locali del sistema operativo, in particolare dalle impostazioni di formato numerico e separatore di elenco nel sistema operativo (Windows).

 Spiegazione tecnica Microsoft Excel, quando salvi un file in formato CSV UTF-8 (delimitato da virgole), in teoria dovrebbe usare sempre la virgola. Tuttavia, Excel può adattare il separatore in base alle impostazioni locali (locale) del sistema operativo, in particolare:

- Se il “Separatore di elenco” nelle impostazioni internazionali è impostato su virgola, Excel userà ,.
- Se invece è impostato su punto e virgola, userà ; anche se hai scelto “CSV delimitato da virgole”.

Questo avviene perché in alcune localizzazioni (come Italia, Germania, Francia), la virgola è usata come separatore decimale, quindi per evitare ambiguità il separatore di campo nei CSV diventa il ;.

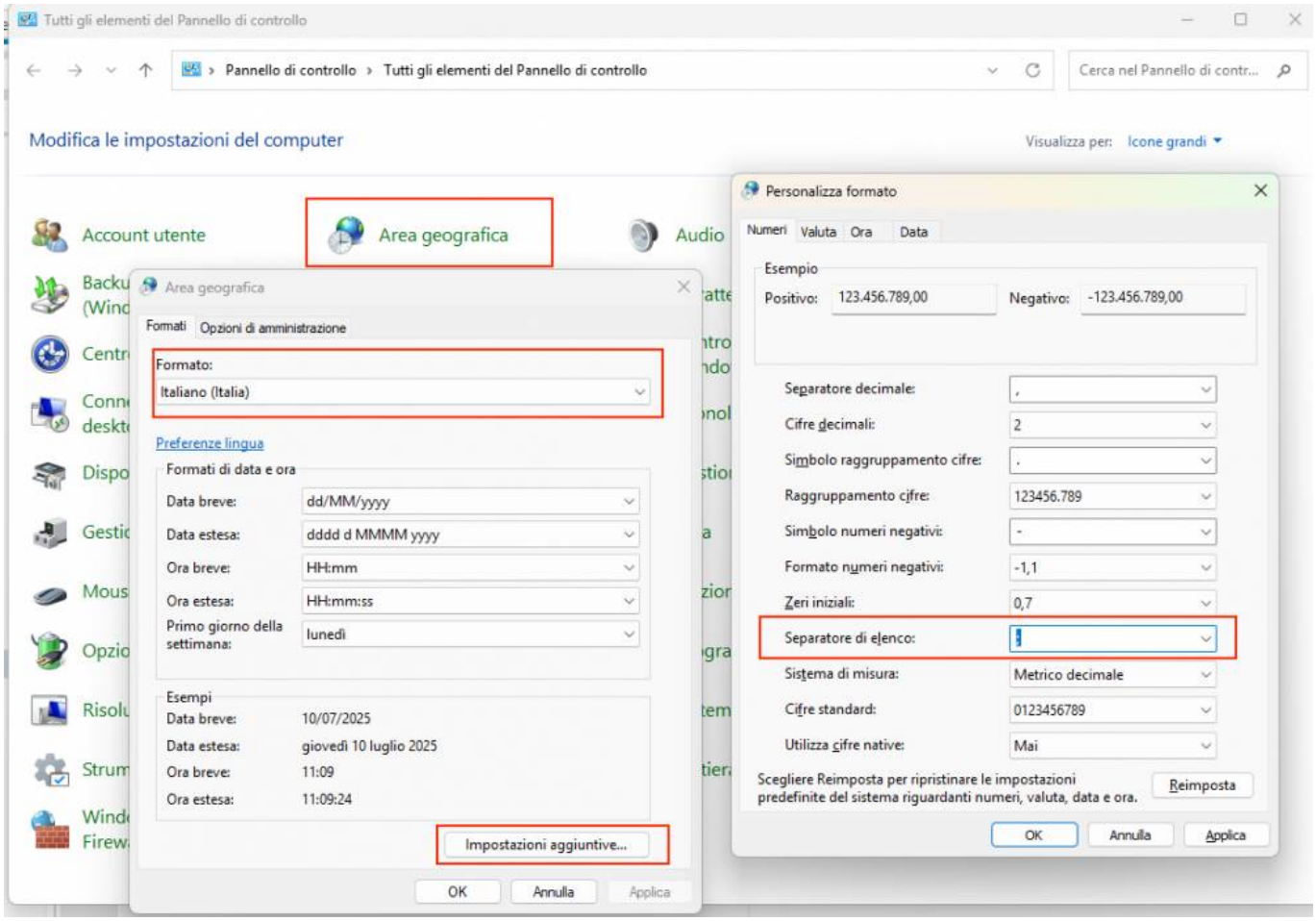
 Come verificare o modificare il comportamento su Windows Apri il Pannello di controllo → Orologio e opzioni internazionali → Area geografica.

Vai alla scheda Formati → clicca su Impostazioni aggiuntive...

Cerca il campo “Separatore di elenco”.

- Se c'è **,**, Excel userà la virgola.
- Se c'è **;**, userà il punto e virgola.

Puoi modificare questo valore temporaneamente se vuoi esportare CSV con un separatore specifico.



Fase 4 - utilizzo delle funzioni di import csv

Importare il file nelle modalità descritte nella sezione dedicata.

SHP

EXPORT SHP (js API)

Esistono attualmente 2 casi d'uso per l'export shape:

1. **CASO 1: Export per aggiornamento**, con **intestazioni Codici** (columnName degli attributi). Usato al fine di essere successivamente reimportato (nell'ambito dei casi d'uso di popolamento massivo, aggiornamento puntuale, riconciliazione dati, etc..)

2. **CASO 2: Export per integrazione con sistemi terzi**, con **intestazioni Personalizzate** (come da remapping). Destinato a produrre file secondo il formato di sistemi esterni, per integrazioni di vario genere.

exportShape()

CASO 1: Export per aggiornamento e CASO 2: Export per integrazione con sistemi terzi

Esportazione SHAPE di tutti gli attributi codificati

Parametri

- **queryParameter**: Object, required
- **grid**: Object, required
- **options**: Object, optional. Può contenere i seguenti parametri:

tipologia	Nome	required	Descrizione breve
Integer	exportType	true	E' un numero intero ed indica il tipo di esportazione. Valori ammessi: 1 (per aggiornamento), 2 (tutti gli attributi)
function	callback	false	Permette di inserire la funzione di callback dopo l importazione
String	codColumn	false	singola colonna o colonne multiple separate da virgola ','. Nome della o delle colonne per le quali, qualora si facesse un importazione tipo update, verrebbero usate come criterio di sovrascrittura. Le colonne se più di una devono essere separate da virgola
Object	columnHeadersMap	false	HashMap per la mappatura dell header dello shape con quelle del database. Vanno inserite soltanto quelle dove il nome è diverso e le si voglia importare. La forma è la seguente: {'nameColumnShp': 'nameColumnDatabase'}
Object	additionalMap	false	HashMap contenente la lista delle colonne non presenti nel file che si voglia comunque popolare durante l inserimento. la forma è {'nameColumnDatabase': 'valore'}, il valore può essere anche preso dinamicamente dalla sessione
String	fileMapping	false	E' il nome del file di mapping se presente nei contenuti statici, deve risiedere nella cartellina SHAPE

Esempi

Esportazione SHAPE di tutti gli attributi codificati

```
exportShape(queryParameter, grid, {exportType: 2});
```

Esportazione SHAPE per AGGIORNAMENTO (la decodifica segue le regole indicate su ciascun attributo)

```
exportShape(queryParameter, grid, {exportType: 1});
```

Esportazione SHAPE per AGGIORNAMENTO (la decodifica segue le regole indicate su ciascun attributo)

```
var callback = function(params){publishGwClassUpdate('gwd_infrastructure')};
```

```
exportShape(
    queryParameter,
    grid,
    {
        exportType: 1,
        fileMapping: '',
        codColumn:'cod_building',
        callback:callback,
        columnHeadersMap:{
'type_ownership':'tipologia_ep','id_building':'id_pod','descr_building':'des
cr_pod','status_building':'status','descr_site':'des_anag','tenant_code':'co
mmessa','cod_site':'cod_anagr','external_code':'pod_cod','cod_building':'cod
_bld','name_building':'nome','cod_building_mc':'cod_mc','id_entity_classific
ation':'entity'
        },
        additionalMap:{
'type_ownership':'Infrastruttura','tenant_code':gwActiveScopesMap.Commessa.s
copeValue
        }
    }
);
```

standardListSelectedAllAttributeExportToShp(queryParameter, grid)

@Deprecated usa exportShape() per maggior controllo

Parametri

- **queryParameter**: Object, required
- **grid**: Object, required

Esportazione SHAPE di tutti gli attributi codificati (senza callback, columnHeadersMap, additionalMap e fileMapping)

```
standardListSelectedAllAttributeExportToShp(queryParameter, grid);
```

IMPORT SHP

importShape()

Parametri

- **grid**: Object, required
- **options**: Object, required. Può contenere i seguenti parametri:

tipologia	Nome	Required	Descrizione breve
Integer	importType	true	è un numero intero ed indica il tipo di importazione. Può assumere il valore 0 o 1 o 2 o 3 o 4 come descritto nella sezione dedicata
function	callback	false	Permette di inserire la funzione di callback dopo l importazione
String	childClassName	false	Nome della classe, qualora si volesse importare i dati in una tabelle di una classe collegata a questa tramite childlist. I record verrebbero gia caricati collegati al record padre dal quale viene lanciata l azione
String	itemId	false	chiave del record da cui si sta lanciando l azione, data.itemDB.pk_column, qualora si volesse importare i dati in una tabelle di una classe collegata a questa tramite childlist. I record verrebbero già caricati collegati al record padre dal quale viene lanciata l azione
String	relationName	false	nome della relazione sulla classe, qualora si volesse importare i dati in una tabelle di una classe collegata a questa tramite childlist. I record verrebbero già
String	codColumn	false	singola colonna o colonne multiple separate da virgola ','. Nome della o delle colonne per le quali, qualora si facesse un importazione tipo update, verrebbero usate come criterio di sovrascrittura. Le colonne se più di una devono essere separate da virgola
Object	columnHeadersMap	false	HashMap per la mappatura dell header dello shape con quelle del database. Vanno inserite soltanto quelle dove il nome è diverso e le si voglia importare. La forma è la seguente {'nameColumnShp': 'nameColumnDatabase'}
Object	additionalMap	false	HashMap contenente la lista delle colonne non presenti nel file che si voglia comunque popolare durante l inserimento. La forma è {'nameColumnDatabase': 'valore'}. Il valore può essere anche preso dinamicamente dalla sessione
String	fileMapping	false	E' il nome del file di mapping se presente nei contenuti statici, deve risiedere nella cartellina SHAPE
String	scriptName	false	Nome dello scripts groovy, CON l'estensione che verrà lanciato al termine del caricamento. Lo script deve essere presente nella cartella dei contenuti statici sottocartella groovy
Integer	handleErrorsType	false	Specifica il comportamento del sistema in presenza di un errore. Se 0,(default) il sistema va avanti segnalando alla fine i records non importati e l errore verificatosi, se 1 blocca l intera importazione se almeno un record non può essere importato
Integer	idProcessImport	false	Se indicato non viene inserito un nuovo record nella tabella contenente l archivio storico delle importazioni bensì aggiorna il record con quel idProcess

tipologia	Nome	Required	Descrizione breve
Integer	itemNotFoundAction	false	Valori ammessi: [0,1]. Default: 0. Valutato solo quando importType vale 2 (update). Quando a 1 fa si che tutti i record vengano aggiornati, impostando la property con nome uguale a itemNotFoundUpdateField con il valore itemNotFoundUpdateValue
String	itemNotFoundUpdateField	false	String, default null. Chiave usata quando itemNotFoundAction==1 && importType==2
String	itemNotFoundUpdateValue	false	String, default null. Valore usato quando itemNotFoundAction==1 && importType==2

Esempi

Importazione minimale dati SHAPE. Viene aperto un dialog per la selezione del file .shp.

Importazione per accodamento.

```
var importType = 0; //alloweb types: 0, 1, 2, 3, 4
importShape(grid, {importType: importType});
```

Importazione per Aggiornamento (TODO DECLINARE MEGLIO).

```
var callback = function(params){publishGwClassUpdate('gwd_infrastructure')};
importShape(
    grid,
    {
        importType: 2,
        fileMapping: '',
        codColumn: 'cod_building',
        callback: callback,
        columnHeadersMap:{
'tipologia_ep':'type_ownership','id_pod':'id_building','descr_pod':'descr_building','status':'status_building','descr_anagr':'descr_site','commessa':'tenant_code','cod_anagr':'cod_site','pod_cod':'external_code','cod_bld':'cod_building','nome':'name_building','cod_mc':'cod_building_mc','entity':'id_entity_classification'
        },
        additionalMap:{
'type_ownership':'Infrastruttura','tenant_code':gwActiveScopesMap.Commessa.scopeValue
        }
    }
);
```

Esempio generico di importazione dei record di una classe figlia dal dettaglio di un record della classe padre, legati da una childlist:

```
importShape(
    grid,
```

```
    {
        importType: importType,
        fileMapping: '',
        gwClassName: 'nome_classe_padre',
        childClassName: 'nome_classe_figlia',
        itemId: data.itemDB.nome_campo_chiave_classe_padre,
        relationName: 'nome_della_relazione',
        codColumn: 'nome_campo_chiave',
        callback: callback,
        columnHeadersMap: {
'nome_campo_nel_database1':'nome_campo_nello_shape1','nome_campo_nel_databas
e2':'nome_campo_nello_shape2',etc…
        },
        additionalMap: { 'campo_nello_shape': 'valore_fisso',
'campo_nello_shape': gwActiveScopesMap.NomeVarSessione.scopeValue }
    }
);
```

Esempio pratico di importazione dei record di una classe figlia dal dettaglio di un record della classe padre, legati da una childlist:

```
var callback = function(params){publishGwClassUpdate('gwd_infrastructure')};
importShape(
    grid,
    {
        importType: 2,
        fileMapping:'',
        gwClassName:'gwd_site',
        childClassName:'gwd_infrastructure',
        itemId:data.itemDB.db_site,
        relationName: 'gwd_site', //'gwd_infrastructure'
        codColumn: 'cod_building',
        callback: callback,
        columnHeadersMap:{
'tipologia_ep':'type_ownership','id_pod':'id_building','descr_pod':'descr_bu
ilding','status':'status_building','descr_anagr':'descr_site','commessa':'te
nant_code','cod_anagr':'cod_site','pod_cod':'external_code','cod_bld':'cod_b
uilding','nome':'name_building','cod_mc':'cod_building_mc','entity':'id_enti
ty_classification'
        },
        additionalMap:{
'type_ownership':'Infrastruttura','tenant_code':gwActiveScopesMap.Commessa.s
copeValue
        }
    }
);
```

Configurazioni Comuni

Tipologie di import

Dove applicabili, sono disponibili 5 tipologie di importazione:

- **importType=0** effettua l'accodamento dei record

```
Esempio: importCSV({gwClassName:'pv_punti_vendita'},{importType: 0});
```

- **importType=1** effettua la sostituzione (delete e insert) dei record sulla base del valore del campo CAMPO_CODICE. (verranno cancellati solo i record del database che possiedono il campo codice uguale a quello dei record presenti nel file importato)

```
Esempio: importCSV({gwClassName:'pv_punti_vendita'},{importType: 1,
codColumn:'cod_punti_vendita'});
```

- **importType=2** effettua l'aggiornamento (update) dei record sulla base del valore del campo CAMPO_CODICE. i campi che vengono aggiornati sono TUTTI i campi presenti nel file di importazione, quindi se si vuole aggiornare, ad esempio, solo un campo, occorre includere solo una colonna intestata a quel campo.\\Se il record non viene trovato, effettua l'accodamento.

```
Esempio: importCSV({gwClassName:'pv_punti_vendita'},{importType: 2,
codColumn:'cod_punti_vendita'});
```

- **importType=3** cancella tutti i record della classe e effettua l'accodamento dei record

```
Esempio: importCSV({gwClassName:'pv_punti_vendita'},{importType: 3});
```

- **importType=4** cancella tutti i record della classe relativamente all'ambito corrente ed effettua l'accodamento il CSV

```
Esempio: importCSV({gwClassName:'pv_punti_vendita'},{importType: 4});
```

Tipo	Operazione	Descrizione
0	Accodamento	insert
1	Sostituzione puntuale	delete + insert
2	Aggiornamento puntuale	update
3	Svuotamento della tabella ed Accodamento	delete all + insert
4	Svuotamento per ambito della tabella ed Accodamento	scoped delete all + insert

Qualora si scelga la tipologia 1 o 2 è obbligatorio il secondo parametro *codColumn*.
codColumn deve essere il nome del campo nella tabella o nel file che identifica in maniera univoca un record; Non è necessario che sia la chiave ma è necessario sia un codice univoco. Questo campo viene utilizzato per ricercare in tabella il record che si sta tentando di importare. Se presente, nel caso 2 viene aggiornato con i nuovi dati, nel caso 1 viene cancellato e reinserito completamente con i dati del record nel file.

Mappatura dei campi

Per importare o esportare i dati dalla tabella al file, è necessaria la corrispondenza tra i campi in tabella e gli header del file affinché i dati di ciascun record finiscano nel campo corrispondete.

Se i nomi delle colonne (field della tabella sul db) coincidono con quelli del file questa mappatura non è necessaria.

Purtroppo non sempre questo è vero, sopratutto nel caso dello shape dove i nomi delle intestazioni delle colonne hanno un massimo di 10 caratteri.

In questo caso è d'obbligo definire la mappatura dei campi, cioè la relazione che intercorre tra i campi del file il cui nome differisce dai nome dei campi in tabella.

La mappatura può avvenire in 3 modalità (ma non tutte le tipologie di importazione/esportazione la supportano), ognuna mutualmente esclusiva con le altre.

Il formato .xlsx, sia in import che in export, non supporta la rimappatura delle colonne.

Matrice di supporto.

Tipo Mappatura	Come si configura	Import. Shp	Import. Csv	Esport. Shp	Esport. Csv
HashMap	parametro columnHeadersMap	Si	Si	Si	No
fileMapping Contenuti Statici	parametro fileMapping	Si	No	Si	No
fileMapping dinamico	Richiesto input	Si	No	Si	No

- **HashMap** Si passa un parametro denominata 'columnHeadersMap' variabile hashmap contenente una lista di chiave\valore dove in fase di esportazione la chiave è il campo nel database e valore il campo nel file e viceversa in fase di importazione.

```
//esportazione
columnHeadersMap:{
  'nome_campo_nel_database1': 'nome_campo_nel_file1',
  'nome_campo_nel_database2': 'nome_campo_nel_file2',
  etc…
}

//importazione
columnHeadersMap:{
  'nome_campo_nel_file1': 'nome_campo_nel_database1',
  'nome_campo_nel_file2': 'nome_campo_nel_database2',
  etc…
}
```

- **file xml presente nei contenuti statici** Si passa un parametro denominata 'fileMapping' contenente il nome (senza estensione) del file presente nei contenuti statici che conterrà la definizione della Mappatura. Il file dovrà risiedere nella cartellina SHAPE dei contenuti statici ed avere questa struttura

```
    <?xml version="1.0"?>
```

```
     <attributes>
      <truncate>true</truncate>
      <attribute><nameFieldShape>NOME_CAMPO_SHAPE1</nameFieldShape>
       <nameFieldDatabase>NOME_CAMPO_NEL_DATABASE1</nameFieldDatabase>
      <defaultValue></defaultValue>
      </attribute>
      <attribute>
      <nameFieldShape>NOME_CAMPO_SHAPE2</nameFieldShape>
       <nameFieldDatabase>NOME_CAMPO_NEL_DATABASE2</nameFieldDatabase>
       <defaultValue></defaultValue>
    </attribute>
   </attributes>
```

- **file xml passato in fase di importazione/esportazione** Nessun parametro aggiuntivo, in fase di importazione\esportazione si dovrà passare dinamicamente volta per volta un file xml con la stessa struttura indicata sopra. In caso di importazione shape questo file dovrà essere passato nello zip dello shape da importare e potrà avere qualsiasi nome. In caso di esportazione Shape il file dovrà essere passato all'interno della form html della maschera che viene aperta all'utente allo voce file di mapping. Tale voce sarà seguita dal pulsante per la scelta del file nel file system. Questo campo della form non è obbligatorio.

```
  fileMapping: ''
```

Priorità di valutazione:

- file di mapping
- contenuti statici
- columnHeadersMap

Campi aggiuntivi nei dati

Ove previsto, in fase di import/export è possibile passare nei dati in transito dal file alla tabella o viceversa, delle informazioni non presenti nella fonte dati di partenza. Praticamente è possibile aggiungere delle colonne in più rispetto alla fonte dati di partenza. Questo è possibile aggiungendo il parametro 'additionalMap' con le colonne da aggiungere e il valore con il quale verranno popolate. In queste colonne, tutti i records assumeranno lo stesso valore. I valori possono essere indicati o come valori fissi oppure con dei valori provenienti da variabili in sessione. GeoWeb infatti, mette a disposizione delle variabile in sessione contenenti, i valori dell'ambito del progetto aperto, il nome dell'utente che si è loggato, il nome del gruppo di appartenza dell'utente etc....

La sintassi da utilizzare è la seguente:

```
additionalMap: {
    'nome_campo': 'valore_fisso',
    'nome_campo': gwActiveScopesMap.NomeVarSessione.scopeValue
}
```

Risultati processi di importazione

Esiste una tabella nello schema dei dati denominata **gw_process_import** dove vengono registrati tutti gli esiti degli import. Verificare che nello schema dei dati sia presente tale tabella che va eventualmente creata con la seguente istruzione:

```
--Postgres
--DROP TABLE IF EXISTS test_gw47data.gw_process_import;

CREATE TABLE IF NOT EXISTS test_gw47data.gw_process_import
(
    id_process_import BIGINT NOT NULL,
    import_date TIMESTAMP WITHOUT TIME zone,
    gw_class CHARACTER VARYING(50) COLLATE pg_catalog."default",
    error_file bytea,
    gw_username CHARACTER VARYING(200) COLLATE pg_catalog."default",
    import_file_blob bytea,
    import_file_name CHARACTER VARYING(100) COLLATE pg_catalog."default",
    import_type INTEGER,
    tot_record INTEGER,
    inserted_record INTEGER,
    updated_record INTEGER,
    error_record INTEGER,
    import_status CHARACTER VARYING(80) COLLATE pg_catalog."default",
    error_file_name CHARACTER VARYING(100) COLLATE pg_catalog."default",
    CONSTRAINT gw_process_import_pkey PRIMARY KEY (id_process_import)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS test_gw47data.gw_process_import
    OWNER TO test_gw47data;

GRANT ALL ON TABLE test_gw47data.gw_process_import TO test_gw47data;
```

Tabella precedente @Deprecated

```
--Postgres
CREATE TABLE gw_process_import
(
  id_process_import INTEGER NOT NULL, -- chiave primaria
  import_date TIMESTAMP WITHOUT TIME zone, -- data e ora di importazione
  gw_class CHARACTER VARYING(50), -- nome della classe su cui viene
effettuata l'importazione
  error_file bytea, -- file di errore
  CONSTRAINT cos_gw_process_import_pk PRIMARY KEY (id_process_import)
)
WITH (OIDS=FALSE);
```

Decodifica dei campi

Parlando dei veri e propri dati caricati o esportati, può accadere che alcuni valori, siano delle codifiche di descrizioni più dettagliate.

GeoWeb, attraverso la configurazione di widget dedicati (come dbcombobox, exsternaltable etc..) va a recuperare la descrizione presente in un tabella esterna presentando all'utente un dato intellegibile nel dettaglio o nella lista di classe.

Quando si esporta o si importa è opportuno indicare cosa stiamo passando in questi campi in fase di importazione o cosa vogliamo in questi campi in fase di esportazione.

Ovvero dobbiamo indicare se si sta lavorando con il codice o con la descrizione.

Alcune volte potrebbe essere comodo passare la descrizione altre il codice.

Questa configurazione è specificata all'interno del file XML di ciascun widget, dove si può specificare se stiamo importando o esportando valori codificati o decodificati.

Il tag di riferiemnto dell'xmld el widget è **importCSVWithoutDecoding**, che può assumere i seguenti valori:

1. **true** inserimento diretto del codice
2. **false** inserimento della descrizione da decodificare

Nota: a dispetto del nome il flag viene valutato anche durante l'esportazione

```
<DBWindowList>
  <width>250</width>
  <height>18</height>
  <maxLength>255</maxLength>
  <required>true</required>
  <readonly>false</readonly>
  <disabled>false</disabled>
  <isDefaultOrderBy>true</isDefaultOrderBy>
  <isDefaultOrderByAscending>true</isDefaultOrderByAscending>
  <isDefaultOrderByOnFieldToShow>true</isDefaultOrderByOnFieldToShow>
  <listCellTextAlign></listCellTextAlign>
  <listCellStyleRules></listCellStyleRules>
  <listCellClass></listCellClass>
  <listCellHeaderStyleRules></listCellHeaderStyleRules>
  <importCSVWithoutDecoding>false</importCSVWithoutDecoding>
  <strQuery>select
pk_tab_tipo_cavo,codice_materiale,description,numero_fibre from
rete_fibra_ottica.tab_tipo_cavo</strQuery>
  <queryClausole></queryClausole>
  <fieldToShow>description</fieldToShow>
  <fieldToStore>pk_tab_tipo_cavo</fieldToStore>
  <initSelValue></initSelValue>
  <columnsLabel>Codice,Materiale,Descrizione,Numero Fibre</columnsLabel>
```

```
<columnsView>pk_tab_tipo_cavo,codice_materiale,description,numero_fibre</col
umnsView>
  <columnsType></columnsType>
  <allowMultipleSelection>false</allowMultipleSelection>
  <multipleSelectionSeparator>,</multipleSelectionSeparator>
</DBWindowList>
```

```
<ComboBox>
  <width>250</width>
  <height>18</height>
  <maxLength>255</maxLength>
  <defaultValue></defaultValue>
  <required>false</required>
  <readonly>false</readonly>
  <disabled>false</disabled>
  <isDefaultOrderBy>true</isDefaultOrderBy>
  <isDefaultOrderByAscending>true</isDefaultOrderByAscending>
  <isDefaultOrderByOnFieldToShow>true</isDefaultOrderByOnFieldToShow>
  <listCellTextAlign></listCellTextAlign>
  <listCellStyleRules></listCellStyleRules>
  <listCellClass></listCellClass>
  <listCellHeaderStyleRules></listCellHeaderStyleRules>
  <importCSVWithoutDecoding>false</importCSVWithoutDecoding>
  <values>
    <string>DPR 462/01</string>
  </values>
  <keys>
    <string>DPR 462/01</string>
  </keys>
  <initSelValue></initSelValue>
</ComboBox>
```

Classi classificate: Attributi Variabili

Un altro concetto molto importante ma che non comporta configurazione particolari per l'utente, è il concetto di classificazione e di attributi variabili. L'importazione e l'esportazione Shape, trattano classi classificate. Se si esporta o si importa una classe classificata il software si preoccupa di portarsi dietro tutti gli attributi variabili. Tutto questo viene gestito dal software in maniera del tutto trasparente all'utente. Attenzione però, qualora si verificano problemi, potrebbe essere uno tra le cause da verificare con attenzione. Gli attributi variabili vengono esportati qualora si lanci l'esportazione da una lista classificata, in caso contrario vengono riportati solo gli attributi comuni a tutte le famiglie. Per ciascun record verrà comunque riportata la famiglia di appartenenza. Deve essere definito l'attributo id_entity_classification come widget classification per la classe e deve essere configurato con importCSVWithoutDecoding impostato a false come segue:

```
<Classification>
  <width>-1</width>
  <height>50</height>
  <listWidth>100</listWidth>
```

```
  <required>false</required>
  <readonly>false</readonly>
  <disabled>false</disabled>
  <hideRelations>false</hideRelations>
  <intermediateChanges>false</intermediateChanges>
  <isDefaultOrderBy>true</isDefaultOrderBy>
  <isDefaultOrderByAscending>true</isDefaultOrderByAscending>
  <isDefaultOrderByOnFieldToShow>true</isDefaultOrderByOnFieldToShow>
  <listCellTextAlign></listCellTextAlign>
  <listCellStyleRules></listCellStyleRules>
  <listCellClass></listCellClass>
  <listCellHeaderStyleRules></listCellHeaderStyleRules>
  <importCSVWithoutDecoding>false</importCSVWithoutDecoding>
  <classReference>gwd_item</classReference>
  <mapPath></mapPath>
</Classification>
```

Il software si preoccupa di mettere come valore nel campo id_entity_classification il codice della famiglia, e gestirlo correttamente tra importazione ed esportazione.

Importazione dal Padre

Fino a questo momento abbiamo trattato azioni di importazione/esportazione a partire dalla lista di una specifica classe geoweb. Quindi azioni:

- **selezionati in lista**
- **selezionati in lista (posto nella Toolbar)**

E' disponibile anche un sistema per importare dati a partire dal dettaglio di classe, con azioni:

- **dettaglio**
- **dettaglio (posto nella Toolbar)**

In altre parole è possibile caricare dati (un numero indefinito di records) di una *classe figlia* (cioè logicamente e gerarchicamente dipendenti da un record della classe *padre*, collegati ad essa tramite una relazione N:1 (n figli per un padre)), dal dettaglio di una *classe padre*.

Precondizione: tra la classe padre e la classe figlia deve essere definita una relazione 1:N nel webadmin, ovvero un collegamento tra due campi delle due tabelle. Si veda a tale proposito la sezione relazioni.

In fase di importazione il campo della tabella della classe figlia che rappresenta il collegamento con la classe padre viene popolato in automatico con il valore della colonna della classe padre del record in cui ci troviamo.

Questo permette di agganciare dinamicamente in fase di importazione già i records alla maschera voluta.

Esempi

```
var callback = function(params){publishGwClassUpdate('nome_classe');};
importShape(
    grid,
    {
        importType: tipo_importazione,
        fileMapping:'',
        gwClassName:'nome_classe_padre',
        childClassName:'nome_classe_figlia',
        itemId:data.itemDB.nome_campo_chiave_classe_padre,
        relationName:nome_della_relazione',
        codColumn:'nome_campo_chiave',
        callback: callback,
        columnHeadersMap:{'nome_campo_nel_database1':'
nome_campo_nello_shape1' ', 'nome_campo_nel_database2':'
nome_campo_nello_shape2',etc…},
        additionalMap:
{'campo_nello_shape':'valore_fisso','campo_nello_shape':gwActiveScopesMap.No
meVarSessione.scopeValue}
    }
);
```

Le tipologie di importazione, il mappaggio dei campi e quant'altro seguono le stesse regole dell'importazione spiegate fino a questo momento.

Shape: gestione dei dati Geometrici

L'importazione/esportazione shape a differenza del csv, puo' essere definita ed utilizzata solamente su classi di tipo Geometrico quindi che riferiscano una tabella con una colonna geometrica. La classe deve essere dichiarata geometrico, e aver definita la sua geomColumn come attributo. Si rimanda alla definizione di classe a questo link. Classi Si puo' definire l'importazione/Esportazione shape anche se la classe non è dichiarata geometrica ma è necessario aver definito almeno l'attributo di tipo (POINT/POLYGON/LINESTRING) sulla colonna geometrica. Se vengono a mancare una di queste due, l'importazione o l'esportazione shape non puo' essere effettuata e se configurata va in errore.

Bisogna far molta attenzione alla configurazione dell'attributo geometrico ma non solo. Durante l'importazione il dato spaziale segue una serie di regole del framework relative al sistema di coordinate che è opportuno sapere e conoscere.

Il sistema di coordinate puo' essere dichiarato in 4 punti:

- shape
- database
- mappa maestro
- fonte dati maestro
- attributo geometrico

Allora, per quanto riguarda lo shape esso può avere qualsiasi sistema di coordinate anche nullo. Per quanto riguarda la configurazione da Maestro, di norma non deve essere specificato il sistema di coordinate sulla fonte dati, ma soltanto sulla mappa. A livello di strumenti di amministrazione di Geoweb invece deve essere specificato il sistema di coordinate sulla mappa che deve coincidere con

quello inserito nella Mappa nella configurazione di Maestro.

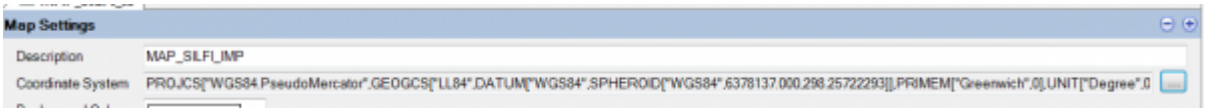
Sempre a livello di Geoweb deve essere definito il sistema di coordinate su ciascun attributo geometrico di ciascuna classe geometrica e deve coincidere con il sistema di coordinate del database.

```
<Point>
  <width>250</width>
  <listWidth>80</listWidth>
  <required>false</required>
  <readonly>false</readonly>
  <isDefaultOrderBy>true</isDefaultOrderBy>
  <isDefaultOrderByAscending>true</isDefaultOrderByAscending>
  <isDefaultOrderByOnFieldToShow>true</isDefaultOrderByOnFieldToShow>
  <listCellTextAlign></listCellTextAlign>
  <listCellStyleRules></listCellStyleRules>
  <listCellClass></listCellClass>
  <listCellHeaderStyleRules></listCellHeaderStyleRules>
  <type>point</type>
  <drawing>false</drawing>
  <drawingType></drawingType>
  <coordinateSystem>4326</coordinateSystem>
  <startingScale>1000.0</startingScale>
</Point>
```

Geoweb infatti può lavorare con un sistema di coordinate differente tra quello definito per il database e quello definito per la mappa, l'importante è settare correttamente il sistema di coordinate nella configurazione del geoweb dagli strumenti di amministrazione e in Maestro. Allora per quanto riguarda l'esportazione, viene restituito un file shape con lo stesso sistema di coordinate del database e quindi quello della tabella esportata. Per quanto riguarda l'importazione invece vengono seguite le seguenti regole. Se il file shape da importare ha un suo sistema di coordinate definito, allora l'importazione esegue una trasformazione della geometria da quel sistema di coordinate a quello del database ovvero al sistema di coordinate definito per l 'attributo geometrico. Se lo shape non ha definito il suo sistema di coordinate si dà per scontato che la geometria presente nello shape è nel sistema di coordinate standard generico 2_d. Se il sistema di coordinate di partenza è differente da quello di arrivo viene effettuata la trasformazione altrimenti si riporta la geometria così come è nello shape. Se non è definito il sistema di coordinate per l'attributo geometrico quindi il sistema di coordinate della destinazione viene fissato quello standard 2_D. In fase di esportazione, vengono esportati soltanto i records la cui geometria non è nulla, gli altri vengono ignorati senza dare l'errore. In fase di importazione, non ci sono problemi perché hanno tutti la geometria.

Definizione del sistema di coordinate nella mappa di Geoweb:

Definizione del sistema di coordinate nella mappa di Maestro:



Supporto ai dati JSON

Linee Guida Sviluppo

Per integrare i flussi import export esistenti, con il supporto del tipo di dato JSON, sono state utilizzate le seguenti linee guida:

- Gestite solo le key di primo livello del JSON
- Per ogni key di primo livello è generata in export una specifica colonna dedicata
- La colonna del file è nominata con una convenzione che concatena il nome del campo JSON, un char separatore specifico (pipe: |), e la key del JSON
- È sempre possibile il remap delle colonne (anche key di un JSON), al fine di integrarsi con sistemi terzi
- Sono supportati gli stessi tipo di dato gestiti dal JSON, come stringhe, numeri interi e decimali, date e boolean
- Si utilizzano, ove possibile, di tutti i vincoli imponibili dalle definizioni delle Famiglie Attributi
- Supporto multi-database: oracle, postgres e mssqlserver.
- Si supportano tutti i meccanismi e casi d'uso precedenti

Key di primo livello del JSON

Ad essere gestite sono solo le key di primo livello. In fase di export, in presenza di un campo JSON, viene eseguita una specifica query (esiste una implementazione per ogni database) che recupera l'unione di tutte le key di primo livello presenti per un campo JSON. A tale proposito:

- Non è rilevante se sul campo JSON è collegata una Famiglia Attributi
- Viene applicato anche a questa query lo stesso filtro, dipendente dal contesto, con cui è stata lanciata l'esportazione
- Si tenta di recuperare il tipo di dato analizzando la prima occorrenza non nulla del valore della key (applicando sempre i filtri di contesto). Esiste una implementazione per ogni database.
- Dal momento che si possono esportare teoricamente record con JSON di struttura totalmente differente tra un record e l'altro (con o meno una Famiglia Attributi differente), c'è da notare che si possono creare inconsistenze non gestibili se con la stessa key in valori JSON differenti è utilizzata per ospitare tipo di dato differenti.
- Il codice è stato ottimizzato per generare al minimo eccezioni generate da inconsistenze. Per esempio una stessa key con valori integer e double su JSON differenti non crea problematiche. Analogamente una stessa key con valori string e boolean su JSON differenti non crea problematiche. Una key con primo valore date o double e successivi valori string potrebbe generare errori in fase di export

Naming dell’intestazione della key

L’header della colonna che si riferisce ad un key JSON, è nominata con una convenzione che concatena il nome del campo JSON, un char separatore specifico (pipe: |) e la key del JSON. Così:

```
[json_field_name]+”|”+[key]
```

Esempio: se il campo JSON si chiama var_attr_jsonb e la key da recuperare è Power (HP) la notazione corretta è var_attr_jsonb|power (HP).

In alcuni casi d’uso, dove il file esportato non è destinato ad essere reimportato, il nome del campo JSON è sostituito dalla label dell’attributo (come per tutti gli altri attributi non JSON). Il char “.” Come divisore nelle intestazioni di colonna è di esclusivo utilizzo del vecchio caso dell’importazione dei figli dal padre (lanciata dal dettaglio padre), che era già sviluppato ed è stato mantenuto.

Da notare che in fase di export i nomi delle intestazioni vengono tutti trasformati in minuscolo. In fase del successivo import, se è possibile recuperare la Famiglia Attributi (in quanto è configurata sopra il JSON ed è effettivamente presente nel tracciato anche il codice della famiglia), essa viene utilizzata per ripristinare esattamente il caso di tutte le key del JSON. Nel caso di JSON semplice, al successivo import, le key del JSON saranno importate tutte come presenti nel file (quindi in minuscolo). Da questo si evince che in fase di implementazione, è possibile, anzi auspicabile, in caso di Famiglia Attributi utilizzare key ben formattate del tutto assimilabili a label (anche se l’utente GW ADMIN può impostare separatamente le key e le relative label). In caso di JSON semplice, per evitare conversioni non volute, è bene fina da subito utilizzare key semplici e minuscole. Si ricorda che comunque, nel caso il JSON semplice vada rappresentato tramite il widget LabelValueJSONVisualizer, è sempre possibile popolare nel JSON stesso la key _metadata, che permette di impostare anche label custom.

Rimappatura colonne con JSON key

Le modalità di mappatura dei campi sono rimaste le medesime. Esempio applicato ad una colonna JSON key:

```
columnHeadersMap:{
‘json_field|key’: 'nome_rimappato_nel_file1',
etc…
}
```

Tipi di dato nel JSON

Il JSON supporta i seguenti tipo di dato:

- String
- Integer

- Double
- Boolean
- Date (indirettamente)

Il tipo di dato Date, non è direttamente supportato dal JSON, ma Geoweb lo supporta mediante stringhe formattate secondo i pattern:

- “yyyy-MM-dd'T'HH:mm:ss.SSS'Z'” ISO-8601 usato internamente a geoweb per salvare date nel JSON sul database
- “yyyy-MM-dd HH:mm:ss” usato per produrre e leggere i file .csv, .xlsx, .shp

I pattern delle date nel JSON sono sempre completi e non hanno mai versioni ridotte come può accadere per i normali attributi di geoweb (che hanno “yyyy-MM-dd HH:mm:ss” o “yyyy-MM-dd”).

A seconda del formato d’esportazione i valori possono essere formattati in maniera differente:

- .csv e .shp
  - Il separatore delle colonne è sempre il punto e virgola (;). Indipendentemente dal Locale dal locale della macchina server.
  - Il separatore dei valori decimali è sempre il punto (.) e mai la virgola
  - Non vengono applicate mai formattazioni particolari per separare le migliaia: spazio ( ) o punto (.)
  - I boolean vengono esportati correttamente “true”, “false”
  - I valori date vengono esportati e presentati come da pattern standard
- .xlsx e .csv convertiti a partire da .xlsx
  - Dipendentemente dal locale della macchina server, Il separatore dei valori decimali può essere il punto (.), per le macchine con localizzazione US, o la virgola (,) per le macchine con localizzazione IT
  - Non vengono applicate mai formattazioni particolari per separare le migliaia: spazio ( ) o punto (.)
  - Dipendentemente dal locale della macchina server, i boolean vengono esportati correttamente maiuscoli e tradotti nella lingua locale: “VERO”/”FALSO” per le macchine con localizzazione IT e “TRUE”/”FALSE” per le macchine con localizzazione US.
  - I valori date vengono esportati come da pattern standard (anche se in .xlsx possono essere formattate)

Geoweb gestisce correttamente l’importazione dei valori a prescindere dal locale della macchina che ha prodotto (o convertito) il file, con alcune forzature:

- Per i boolean l’unico locale differente da US supportato è IT. Sono comunque sia validi tutti i locali che rendono i boolean come “TRUE”/”FALSE”

Policy di importazione

Di fronte ad un file di import che presenta nuove colonne che si riferiscono a key non esistenti in alcun JSON, geoweb attua la seguente regola:

- In caso di JSON con Famiglia Attributi (cioè sul quale c’è un PropertySetWidget verranno persistite SOLO le key per le quali esiste la definizione della relativa proprietà nella Famiglia Attributi.
- In caso di JSON semplice tutte le key e relativi valori sono salvati nel JSON del record importato.

Quindi attualmente si delineano due casi d’uso ben distinti.

Il primo è quello di esercizio, dove il SM (od il GW ADMIN) censiscono e definiscono preventivamente le Famiglie Attributi e le relative proprietà. La possibilità di introdurre dati corrotti tramite import da parte dell’utente è ridotta. Il secondo è quello in fase di popolamento dati iniziale o prototipazione del sistema, dove c’è totale libertà di import dei dati al fine di popolare il dato JSON. Nulla vieta che in una fase successiva su quel campo possa essere definita una Famiglia Attributi, magari costruita specificatamente in base ai valori effettivamente presenti.

Trigger di classe .groovy Durante le importazioni naturalmente girano i trigger applicativi configurati per la classe. Per i tipi di dato JSON è predisposto un utilizzo map-like delle key nel groovy. I valori delle key sono quindi accessibili con notazione puntata.

Esempi:

```
valuesMap.data_json_b.key_1 = “value one”;
…
valuesMap.data_json_b._metadata.groups.group_1.label = “Custom Label”;
```

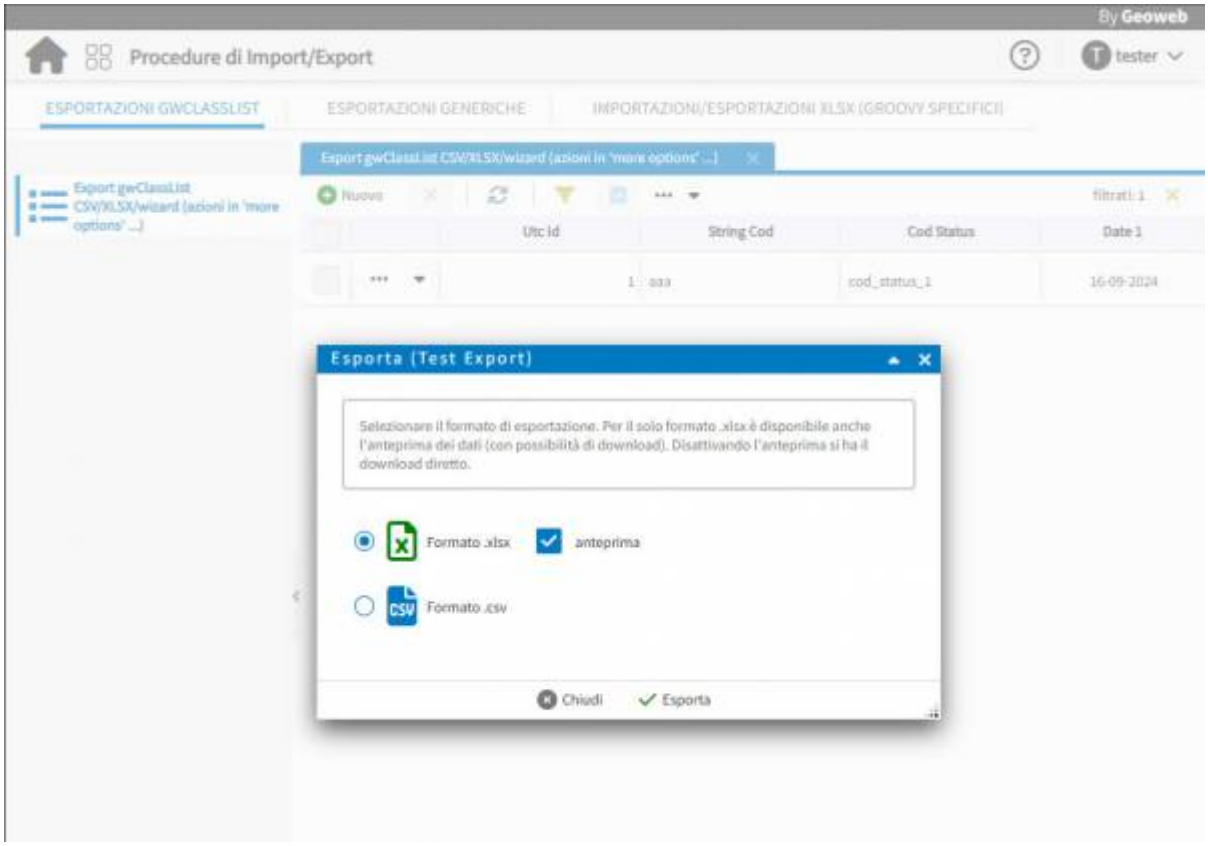
Wizard Export gwClassList

(dalla versione 4.7.4)

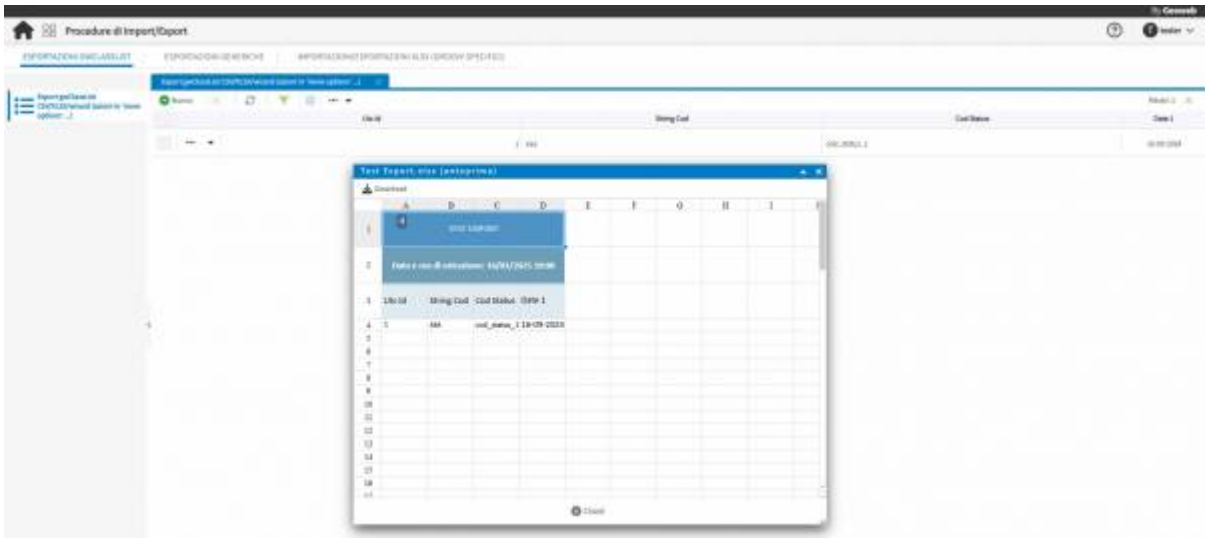
necessario plugin **gw-csv-connector** version **2.0.2**

E' disponibile una api js per avviare un mini *wizard* di esportazione da tutte le gwClassList. Basterà implementare una *gwAction* '*Selezionati in Lista*'/'*Selezionati* in Lista (posta nella toolbar)', che utilizzi l'api js **openExportWizardFromGwClassList()**.

All'esecuzione dell'azione verrà aperto un piccolo floatingPane, che permette il download diretto nei formati .csv/.xlsx. Per il solo formato .xlsx verrà presentata un'anteprima in floatingPane (c'è un checkBox per escludere questo comportamento ed andare al download diretto).



Dalla preview .xlsx è disponibile il download.



openExportWizardFromGwClassList()

Lavora con i record selezionanti in lista. Se l'azione non è di tipo 'selezionati in lista', con queryParameter gia calcolato e disponibile come parametro, queryParameter viene ricalcolato in base alla selezione corrente.

Parametri

- **grid**: Object, required
- **options**: Object, optional, usata per ospitare gli stessi parametri delle options per ospitare i parametri generici del floatingPane wizard. Nel caso di export .csv le options sono riversate

nelle options della api js exportCSVListSelectedAllAttributes(grid, options) (usata internamente). In caso di export .xlsx le options non sono utilizzate dai metodi interni (openGwXlsxViewerFloatingPane(params) per la preview e standardListSelectedAllAttributeExportToExcel(queryParameter, grid) per l'export diretto). In generale è possibile passare un parametro queryParameter con i filtri, che evita che venga recuperato in automatico quello corrente in base alla selection della grid. Questo è utile nel caso delle azioni 'selezionati in lista', che devono lavorare sui record selezionati, e dove queryParameter è gia calcolato.

Esempio

```
var options = {};
openExportWizardFromGwClassList(grid, options);
```

Esempio - selezionati in lista

```
var options = {queryParameter: queryParameter};
openExportWizardFromGwClassList(grid, options);
```