''openUploadUI()''

Panoramica

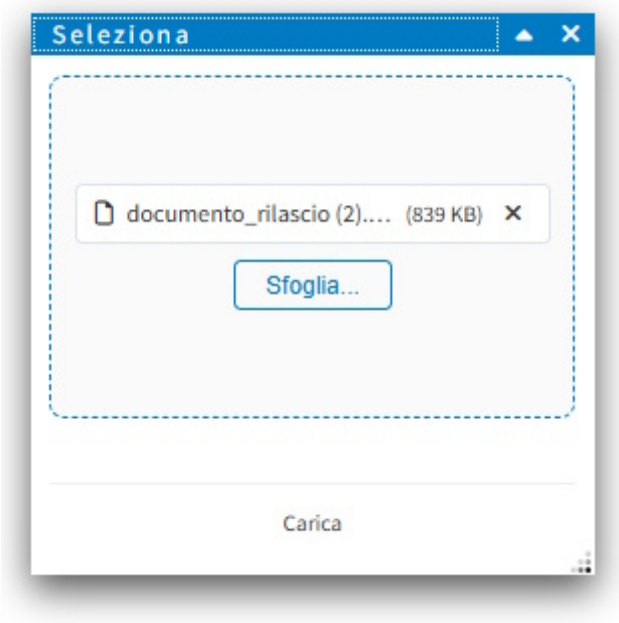
`openUploadUI()` apre un'interfaccia grafica per il caricamento di file che invia i file selezionati al server, dove vengono elaborati da uno script **Groovy** specificato dallo sviluppatore.

L'UI offre:

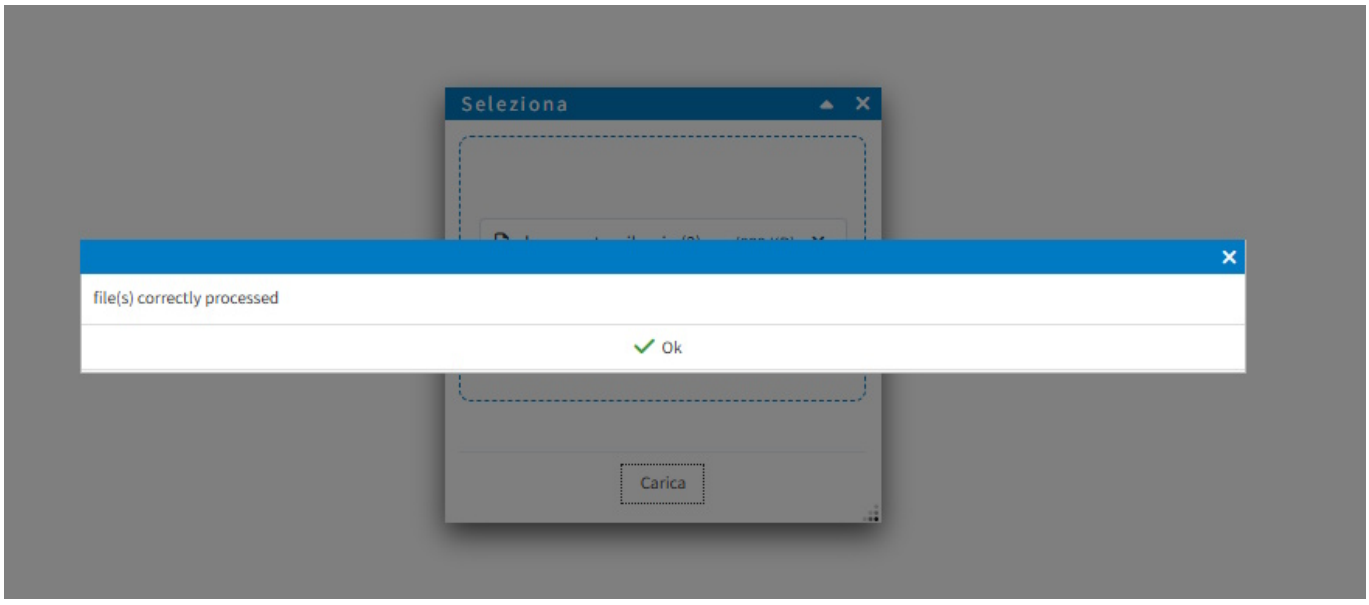
- **Area drag & drop** con selezione via file picker dell'OS
- **Lista dei file selezionati** con possibilità di rimozione singola
- **Validazione client-side** su estensione e dimensione totale
- **Upload automatico** oppure su click di un pulsante “Carica”
- Apertura in **FloatingPane** (default) oppure in un **contenitore esistente**
- **Notifiche** via dialog, MessageToaster o callback personalizzato

UI minimale, nessun parametro:

UI che mostra la selezione corrente



Notifica standard si avvenuto caricamento in base al messaggio ritornato dal groovy



Firma della funzione

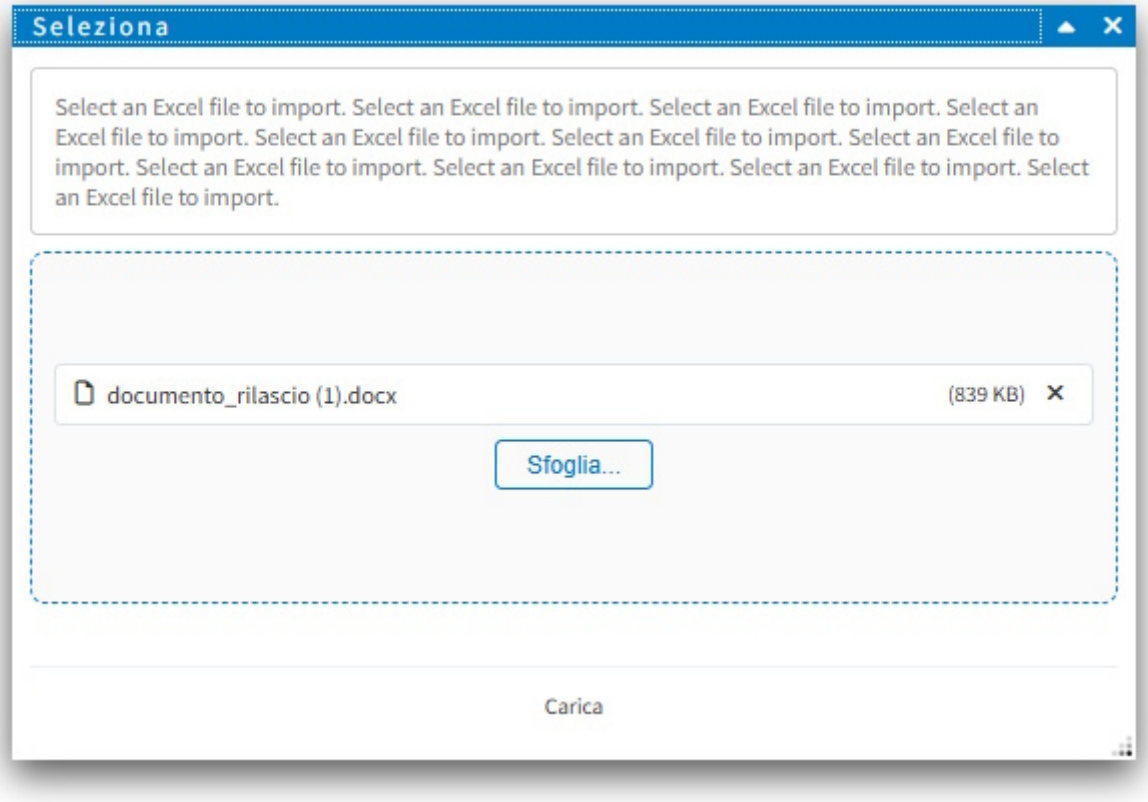
```
openUploadUI(scriptName, options)
```

Parametro	Tipo	Obbligatorio	Descrizione
`scriptName`	`String`	Sì	Nome in notazione dot del Groovy script da eseguire sul server (es. “`mypackage.ImportScript`”). L'estensione `.groovy` viene aggiunta automaticamente.
`options`	`Object`	No	Oggetto di configurazione opzionale. Tutti i sotto-parametri sono opzionali.

Parametri di ''options''

Testo e istruzioni

Parametro	Tipo	Default	Descrizione
`instructions`	`String`	`null`	Testo di istruzioni mostrato all'utente in un'area stilizzata (`gwInstructionsArea`) sopra la drop zone. Accetta HTML.



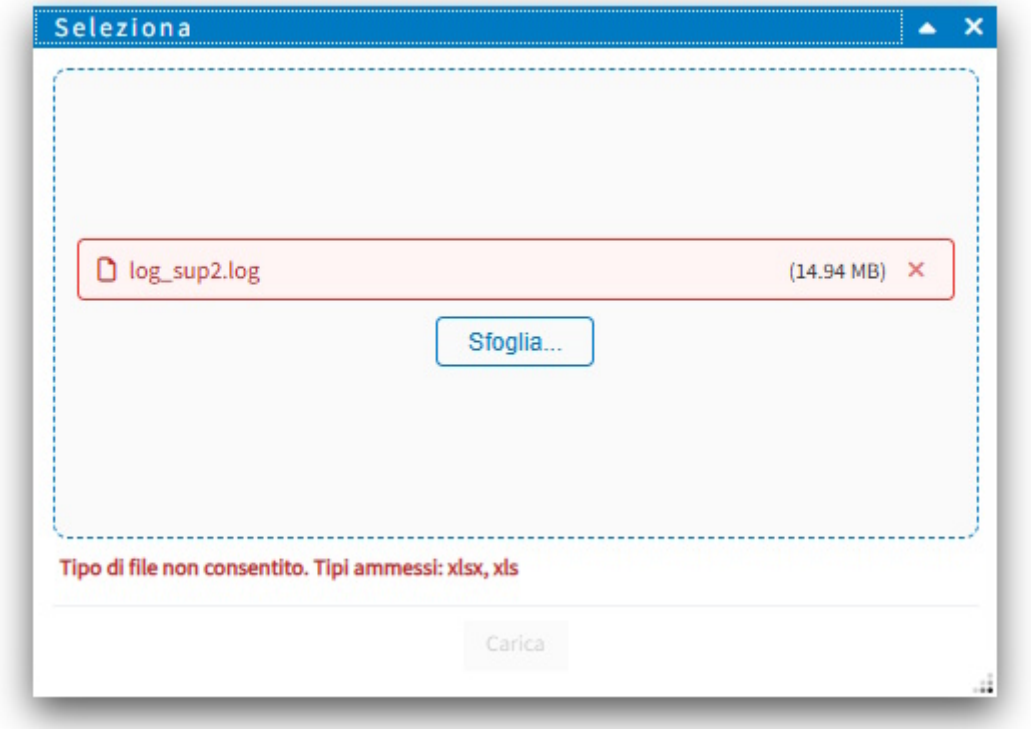
Comportamento upload

Parametro	Tipo	Default	Descrizione
`autoupload`	`Boolean`	`false`	Se `true`, l'upload parte automaticamente non appena i file vengono selezionati o trascinati. Il pulsante “Carica” non viene mostrato.
`multiple`	`Boolean`	`false`	Se `true`, permette la selezione e il caricamento di più file contemporaneamente. Se `false` e l'utente trascina più file, viene preso solo il primo.
`maxUploadSizeMB`	`Number`	`null`	Dimensione massima totale in MB (somma di tutti i file selezionati). Se superata, il pulsante “Carica” viene disabilitato e viene mostrato un messaggio di errore. `null` = nessun limite.
`allowedExtensions`	`String[]`	`null`	Array di estensioni consentite senza il punto iniziale (es. `['pdf', 'xlsx']`). La restrizione viene propagata anche al file picker dell'OS tramite l'attributo `accept`. `null` = tutti i tipi consentiti.

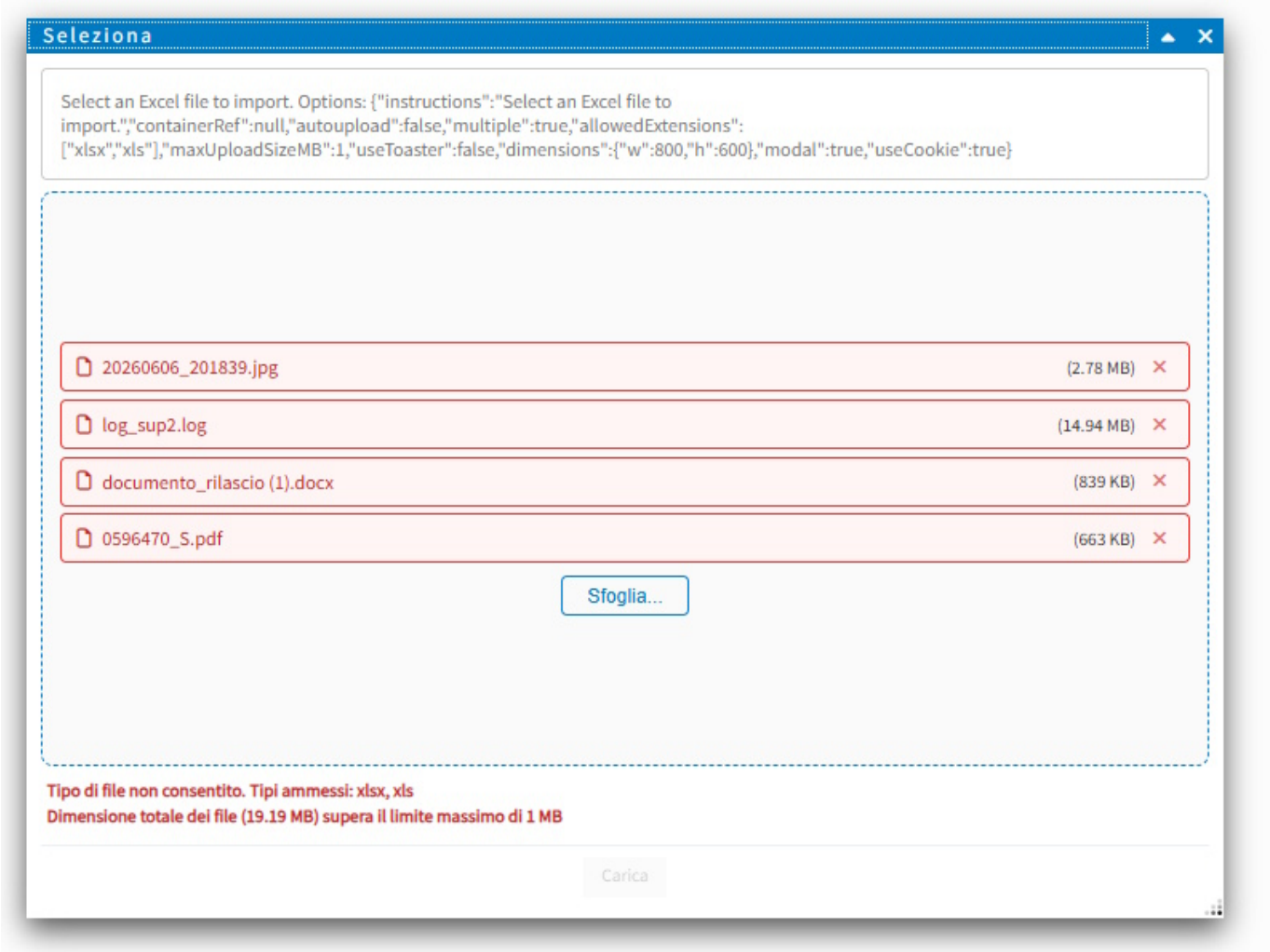
Posizionamento UI

Parametro	Tipo	Default	Descrizione
`containerRef`	`String | Widget | Node`	`null`	Se assente, l'UI si apre in un FloatingPane. Se presente, l'UI viene inserita all'interno del contenitore indicato. Valori accettati: widget id (String), DOM node id (String), oggetto widget Dojo (ha `.domNode`), nodo DOM diretto.
`dimensions`	`Object`	`{w:600, h:600}`	Dimensioni del FloatingPane in pixel. Ignorato se `containerRef` è presente. Formato: `{w: 800, h: 500}`.
`modal`	`Boolean`	`false`	Se `true`, il FloatingPane apre in modalità modale. Ignorato se `containerRef` è presente.
`useCookie`	`Boolean`	`false`	Se `true`, la posizione e le dimensioni del FloatingPane vengono salvate in localStorage e ripristinate alla successiva apertura. Ignorato se `containerRef` è presente.

Esempio di validazione fallita



Esempio con utilizzo esteso dei parametri:



Notifiche e callback

Parametro	Tipo	Default	Descrizione
`useToaster`	`Boolean`	`false`	Se `true`, la notifica di successo viene mostrata via `messageToaster` invece di `showOkDialog()`. Gli errori usano sempre `showErrorDialog()` salvo `errorCallback` personalizzato.
`callback`	`Function`	`null`	Funzione chiamata dopo una risposta di successo. Riceve come unico argomento il `JsonServerResponse` completo. Se assente, viene usata la notifica predefinita (dialog o toaster).
`errorCallback`	`Function`	`null`	Funzione chiamata in caso di errore. Riceve `{success: false, description: String}`. Se assente, viene chiamata `showErrorDialog()`.

Parametri aggiuntivi per il Groovy

Parametro	Tipo	Default	Descrizione
`parameters`	`Object`	`null`	Mappa di parametri aggiuntivi da passare allo script Groovy nel binding. Vedi sezione Variabili disponibili nel Groovy.

Comportamento al successo

Modalità apertura	Comportamento on success
FloatingPane + `showOkDialog` (default)	Il FloatingPane si chiude quando l'utente clicca OK nel dialog.
FloatingPane + `useToaster: true`	Il FloatingPane si chiude immediatamente; poi compare il toaster.
FloatingPane + `callback` personalizzato	Il FloatingPane si chiude immediatamente dopo il callback.
containerRef + `showOkDialog` (default)	L'UI si reimposta allo stato iniziale (vuota) quando l'utente clicca OK.
containerRef + `useToaster: true`	L'UI si reimposta allo stato iniziale immediatamente; poi compare il toaster.
containerRef + `callback` personalizzato	L'UI si reimposta allo stato iniziale immediatamente dopo il callback.

Variabili disponibili nel Groovy

Lo script Groovy riceve nel `Binding` le seguenti variabili:

Specifiche dell'upload

Variabile	Tipo	Descrizione
`file`	`MultipartFile`	Il primo file caricato (o `null` se nessun file).
`files`	`MultipartFile[]`	Array di tutti i file caricati.
`parametersMap`	`HashMap<String, Object>`	Mappa dei parametri aggiuntivi passati tramite `options.parameters`.
`parameters`	`HashMap<String, Object>`	Alias di `parametersMap`.
`<chiave>`	`Object`	Ogni entry di `parameters` è anche disponibile come variabile individuale nel binding (es. `parameters: {userId: 42}` → la variabile `userId` vale 42).
`jsonServerResponse`	`JsonServerResponse`	L'oggetto di risposta. Disponibile per lettura; `success` e `description` vengono applicati dal Map restituito dallo script.

Variabili di sessione (aggiunte da ''addSessionObjectInfo'')

Variabile	Tipo	Descrizione
`gw_activeUser`	`String`	Utente attivo.
`gw_activeGroup`	`String`	Gruppo attivo.
`gw_activeScopes`	`List`	Scope attivi (definizioni).
`gw_activeScopeList`	`List<Map>`	Scope attivi (forma leggibile).
`gw_projectName`	`String`	Nome del progetto.
`gw_solutionCode`	`String`	Codice soluzione dalla configurazione.

Servizi (aggiunti automaticamente da ''runScript'')

Variabile	Tipo	Descrizione
`services`	`HashMap`	Tutti i servizi registrati nel `GwRegistry`.
`log`	`Logger`	Logger Log4j per il Groovy script.
`applicationContext`	`ApplicationContext`	Contesto Spring.
`environment`	`Environment`	Variabili d'ambiente Spring.
ogni servizio	vario	Ogni servizio è anche disponibile come variabile individuale.

Contratto della risposta

Il controller legge **esclusivamente il Map restituito** dallo script per determinare successo e messaggio:

Chiave	Tipo	Descrizione
`success`	`Boolean`	`true` → operazione riuscita; `false` → fallimento
`message`	`String`	Messaggio mostrato all'utente (nel dialog o nel toaster)

Lo script può aggiungere dati extra alla risposta tramite `jsonServerResponse.addResponseObject("key", value)`, accessibili nel callback JS come `resp.responseHashMap.key`.

<note warning> **Attenzione:** se lo script restituisce `null` o un tipo non-Map, il controller lancia una `RuntimeException` e l'upload viene considerato fallito. </note>

Template standard di Groovy script

```
// Script in: mypackage/ImportScript.groovy  →  scriptName:
"mypackage.ImportScript"

def result = [:]
def success = false
def message = null

try {
    // Variabili disponibili nel binding:
    //   file           {MultipartFile}   – primo file caricato (o null se
assente)
    //   files          {MultipartFile[]} – tutti i file caricati
    //   parametersMap  {Map}             – parametri extra inviati dal JS
(alias: parameters)
    //   <chiave>       {Object}          – ogni entry di parametersMap è
disponibile individualmente
    //   gw_activeUser, gw_activeGroup, gw_projectName, gw_projectPath, ...
    //   services, log, applicationContext, environment, ...
```

```
    def parameterValue = parametersMap.parameter_name

    // ... elaborazione ...
    byte[] bytes = file.getBytes()
    // ...

    success = true
    message = 'file(s) correctly processed'

} catch (Exception e) {
    log.error(e.getMessage(), e)
    success = false
    message = 'ko'
} finally {
    result.success = success
    result.message = message
}

return result
```

Esempi di utilizzo

1 — Uso minimo (solo ''scriptName'')

```
// Apre il FloatingPane default 600×600.
// L'utente seleziona un file e clicca "Carica".
// Al successo compare showOkDialog con il messaggio del server.

openUploadUI('mypackage.SimpleImport');
```

---

2 — ''instructions''

```
// Mostra un'area di testo con istruzioni sopra la drop zone.

openUploadUI('mypackage.SimpleImport', {
    instructions: 'Carica il file di importazione in formato Excel.<br>' +
                  'Il file non deve superare le 1000 righe.'
});
```

---

3 — ''autoupload''

```
// L'upload parte automaticamente non appena il file viene
```

```
selezionato/trascinato.
// Il pulsante "Carica" non compare.

openUploadUI('mypackage.QuickProcess', {
    autoupload: true
});
```

---

4 — ''multiple''

```
// Permette la selezione e il caricamento di più file in una sola chiamata.
// Il server riceve files[] con tutti i file.

openUploadUI('mypackage.BatchImport', {
    multiple: true
});
```

---

5 — ''maxUploadSizeMB''

```
// Impedisce l'upload se la dimensione totale supera 5 MB.
// Il bottone "Carica" viene disabilitato e compare un messaggio
localizzato.

openUploadUI('mypackage.DocumentUpload', {
    maxUploadSizeMB: 5
});
```

---

6 — ''allowedExtensions''

```
// Accetta solo file PDF e Word. La restrizione vale sia per il drag&drop
// sia per il file picker dell'OS (attributo accept).

openUploadUI('mypackage.DocumentImport', {
    allowedExtensions: ['pdf', 'doc', 'docx']
});
```

---

7 — ''containerRef'' (widget id)

```
// L'UI viene inserita nel widget Dojo con id "myContentPane"
// invece di aprire un FloatingPane.
```

```
openUploadUI('mypackage.EmbeddedUpload', {
    containerRef: 'myContentPane'
});
```

---

8 — ''containerRef'' (nodo DOM)

```
// L'UI viene inserita direttamente in un nodo DOM.

var container = document.getElementById('uploadArea');
openUploadUI('mypackage.EmbeddedUpload', {
    containerRef: container
});
```

---

9 — ''dimensions''

```
// Il FloatingPane viene aperto con dimensioni personalizzate.

openUploadUI('mypackage.LargeImport', {
    dimensions: { w: 800, h: 500 }
});
```

---

10 — ''modal''

```
// Il FloatingPane viene aperto in modalità modale:
// l'utente non può interagire con il resto della pagina finché non chiude
il pannello.

openUploadUI('mypackage.CriticalImport', {
    modal: true
});
```

---

11 — ''useCookie''

```
// La posizione e le dimensioni del FloatingPane vengono salvate in
localStorage.
// Alla successiva apertura il pannello si riposiziona dove era stato
lasciato.

openUploadUI('mypackage.RecurringImport', {
    useCookie: true
```

```
});
```

---

12 — ''useToaster''

```
// Il messaggio di successo appare nel MessageToaster invece che in un
dialog.
// Utile quando si vuole una notifica non bloccante.

openUploadUI('mypackage.SilentImport', {
    useToaster: true
});
```

---

13 — ''parameters''

```
// Passa parametri aggiuntivi al Groovy script.
// Nel binding saranno disponibili come:
//   parametersMap["targetTable"], parameters["targetTable"]  → "anagrafica"
//   targetTable  → "anagrafica"   (variabile individuale)
//   overwrite    → true           (variabile individuale)

openUploadUI('mypackage.TableImport', {
    parameters: {
        targetTable: 'anagrafica',
        overwrite:   true,
        userId:      42
    }
});
```

Lato Groovy:

```
log.info("Importazione in tabella: ${targetTable}, overwrite: ${overwrite}")
// oppure
log.info("Tabella: ${parametersMap.targetTable}")
```

---

14 — ''callback''

```
// Callback personalizzato al successo: riceve il JsonServerResponse
completo.
// Il FloatingPane si chiude ugualmente dopo il callback.

openUploadUI('mypackage.ImportWithResult', {
    callback: function(resp) {
        var importedCount = resp.responseHashMap.result.importedCount;
```

```
        showOkDialog({ message: 'Importati ' + importedCount + ' record.'
});
        // aggiorna la datagrid aperta
        refreshDatagrid();
    }
});
```

---

15 — "errorCallback"

```
// Gestione personalizzata degli errori di rete o di script.

openUploadUI('mypackage.CriticalImport', {
    errorCallback: function(err) {
        console.error('Upload fallito:', err.description);
        showErrorDialog('Errore durante il caricamento: ' +
err.description);
        // logica di ripristino personalizzata
        rollbackLocalState();
    }
});
```

---

16 — Tutti i parametri combinati

```
openUploadUI('reportistica.ImportazioneExcel', {

    // Istruzioni per l'utente
    instructions: 'Carica il file Excel con i dati da importare.<br>' +
                  'Sono ammessi solo file .xlsx e .xls, max 10 MB.',

    // Selezione e validazione
    multiple:           false,
    allowedExtensions: ['xlsx', 'xls'],
    maxUploadSizeMB:    10,
    autoupload:         false,

    // Posizionamento: FloatingPane personalizzato
    containerRef:       null,
    dimensions:         { w: 750, h: 550 },
    modal:              true,
    useCookie:          false,

    // Parametri extra per il Groovy
    parameters: {
        projectCode:  gwProjectName,
        targetEntity: 'Dipendenti',
        dryRun:       false
```

```
    },

    // Notifica di successo non bloccante
    useToaster: false,

    // Callback di successo personalizzato
    callback: function(resp) {
        var result = resp.responseHashMap.result;
        showOkDialog({
            message: 'Importazione completata.<br>' +
                     'Record inseriti: <b>' + result.insertedRows +
'</b><br>' +
                     'Errori: <b>' + result.errorRows + '</b>'
        });
        topic.publish('myApp/dataChanged', { entity: 'Dipendenti' });
    },

    // Gestione errori personalizzata
    errorCallback: function(err) {
        showErrorDialog('Importazione fallita: ' + err.description);
    }
});
```

Casi d'uso

CU-01 — Importazione dati da file Excel

L'utente carica un file `.xlsx` contenente anagrafiche. Il Groovy legge il file riga per riga (con Apache POI, già disponibile tramite i servizi nel binding) e inserisce o aggiorna i record nel database.

```
openUploadUI('import.AnagraficaImport', {
    instructions:      'File Excel con le colonne: Cognome, Nome, CF,
Email.',
    allowedExtensions: ['xlsx', 'xls'],
    maxUploadSizeMB:   2,
    useToaster:        true,
    callback: function(resp) {
        datagridAnagrafica.refresh();
    }
});
```

CU-02 — Caricamento allegato a una scheda (embedded)

L'UI viene visualizzata all'interno di un pannello del dettaglio scheda già aperto, senza aprire un

nuovo FloatingPane. Al successo, la lista allegati viene aggiornata.

```
openUploadUI('allegati.CaricaAllegato', {
    containerRef:      dijit.registry.byId('attachmentsTab'),
    allowedExtensions: ['pdf', 'jpg', 'png', 'docx'],
    maxUploadSizeMB:   20,
    parameters:        { itemId: currentItemId, className: 'Pratica' },
    callback: function(resp) {
        refreshAttachmentsList();
    }
});
```

CU-03 — Upload multiplo di immagini con auto-invio

L'utente trascina più immagini contemporaneamente. L'upload parte senza necessità di cliccare alcun pulsante. Il Groovy ridimensiona e archivia le immagini.

```
openUploadUI('media.ImageUploader', {
    instructions:      'Trascina le immagini o usa il pulsante Sfoglia.
Formati ammessi: JPG, PNG, WEBP.',
    multiple:          true,
    autoupload:        true,
    allowedExtensions: ['jpg', 'jpeg', 'png', 'webp'],
    maxUploadSizeMB:   50,
    dimensions:        { w: 700, h: 480 },
    useToaster:        true
});
```

CU-04 — Importazione configurazione da JSON

Un amministratore carica un file di configurazione. Il Groovy valida il JSON, applica la configurazione e restituisce un report dettagliato. L'UI si apre in modalità modale per impedire operazioni concorrenti.

```
openUploadUI('admin.ConfigImport', {
    instructions:      "Carica il file di configurazione esportato.
Attenzione: l'operazione è irreversibile.",
    allowedExtensions: ['json'],
    maxUploadSizeMB:   1,
    modal:             true,
    parameters:        { environment: 'production', validateOnly: false },
    callback: function(resp) {
        var report = resp.responseHashMap.result;
        showOkDialog({
            message: 'Configurazione applicata.<br>Voci aggiornate: ' +
report.updatedKeys
        });
```

```
    },
    errorCallback: function(err) {
        showErrorDialog('Configurazione non valida: ' + err.description);
    }
});
```

CU-05 — Sostituzione di un documento con versionamento

L'utente rimpiazza un documento esistente. Il Groovy archivia la versione precedente e salva quella nuova, restituendo metadati sulla versione creata.

```
openUploadUI('documenti.SostituisciDocumento', {
    allowedExtensions: ['pdf'],
    maxUploadSizeMB:   30,
    useCookie:         true,
    parameters: {
        documentId:  currentDocumentId,
        keepHistory: true
    },
    callback: function(resp) {
        var meta = resp.responseHashMap.result;
        messageToaster.setContent(
            'Versione ' + meta.versionNumber + ' salvata.', 'message'
        );
        messageToaster.show();
        reloadDocumentDetail(currentDocumentId);
    }
});
```

CU-06 — Importazione batch da più file CSV

L'ufficio carica più file CSV (uno per categoria) in un'unica operazione. Il Groovy li elabora in sequenza e produce un report consolidato.

```
openUploadUI('batch.MultiCSVImport', {
    instructions:      'Seleziona tutti i file CSV da importare in una volta
sola.',
    multiple:          true,
    allowedExtensions: ['csv'],
    maxUploadSizeMB:   25,
    dimensions:        { w: 700, h: 500 },
    parameters:        { separator: ';', encoding: 'UTF-8' },
    callback: function(resp) {
        var r = resp.responseHashMap.result;
        showOkDialog({
            message: 'Elaborati ' + r.filesProcessed + ' file.<br>' +
```

```
                    'Righe importate: ' + r.totalRows + '<br>' +
                    'Errori: ' + r.totalErrors
        });
        refreshReportDashboard();
    }
});
```

CU-07 — Validazione preventiva (dry run)

Prima di un import definitivo, l'utente può eseguire una validazione “dry run” che non scrive sul database ma restituisce l'elenco degli errori trovati.

```
function apriValidazione(isDryRun) {
    openUploadUI('import.ValidatedImport', {
        instructions: isDryRun
            ? 'Modalità VERIFICA: nessun dato verrà scritto nel database.'
            : 'Modalità IMPORTAZIONE: i dati verranno scritti nel
database.',
        allowedExtensions: ['xlsx'],
        maxUploadSizeMB:   5,
        parameters: {
            dryRun:      isDryRun,
            targetTable: 'Contratti'
        },
        callback: function(resp) {
            var r = resp.responseHashMap.result;
            if (isDryRun) {
                showOkDialog({
                    message: 'Verifica completata.<br>' +
                             'Righe valide: ' + r.validRows + '<br>' +
                             'Righe con errori: ' + r.errorRows
                });
            } else {
                messageToaster.setContent('Importati ' + r.insertedRows + '
record.', 'message');
                messageToaster.show();
            }
        }
    });
}

// Uso:
apriValidazione(true);   // solo verifica
apriValidazione(false);  // importazione reale
```

Note tecniche

Formato della risposta del server

Il controller restituisce sempre un `JsonServerResponse` in formato JSON:

```
{
  "success": true,
  "description": "Operazione eseguita con successo",
  "errorDetailList": [],
  "responseHashMap": {
    "result": { ... }
  }
}
```

Il campo `result` contiene il valore di ritorno dello script Groovy.

Dimensione massima upload lato server

La dimensione massima lato server è configurata in `dispatcher-servlet.xml` tramite `CommonsMultipartResolver`. Il parametro `maxUploadSizeMB` in `openUploadUI` è una **validazione client-side** che si aggiunge (non sostituisce) al limite server.

Compatibilità con ''containerRef''

Quando si usa `containerRef`, l'UI occupa il 100% dello spazio disponibile nel contenitore (`width: 100%; height: 100%`). Il contenitore deve avere una dimensione definita (es. `height` esplicita o essere un `ContentPane` Dojo con layout calcolato).

Variabili globali richieste

La funzione dipende dalle seguenti variabili globali definite dalla pagina JSP:

Variabile	Provenienza	Uso
`gwContextPath`	`i18nGlobalMessages.jsp`	Base URL delle richieste
`gwProjectName`	`project.jsp`	Path variable del controller
`selectLabel`	`i18nGlobalMessages.jsp`	Titolo del FloatingPane
`uploadLabel`	`i18nGlobalMessages.jsp`	Label del pulsante “Carica”
`gwUploadDropFilesLabel`	`i18nGlobalMessages.jsp`	Testo della drop zone
`gwUploadBrowseLabel`	`i18nGlobalMessages.jsp`	Label pulsante “Sfoglia”
`gwUploadSizeLimitExceededLabel`	`i18nGlobalMessages.jsp`	Messaggio di errore dimensione
`gwUploadInvalidExtensionLabel`	`i18nGlobalMessages.jsp`	Messaggio di errore estensione
`generalErrorMessage`	`i18nGlobalMessages.jsp`	Messaggio di errore generico

Note

Issue di riferimento: #1824
File sorgente JS: `gw-commons-web/…/debug/js/gwCommons.js`
Controller: `gw-webclient/…/controller/commons/UploadToGroovyController.java`
Endpoint: `POST /{projectName}/uploadToGroovy`