Pubblicazione Eventi (pattern publish/subscribe)

- Nozioni generali
- Elenco function API
- Altri eventi di interesse sottoscrivibili
- Trick sottoscrizione eventi

Nozioni generali

In Geoweb esiste un meccanismo di gestione di propagazione degli eventi lato client, che si appoggia sulla funzionalità di *publish/subscribe* messa a disposizione dal modulo *dojo/topic* di Dojo (ndr: Dojo è il framework javascript su cui è basato Geoweb). Questa funzionalità è usata internamente a Geoweb per sincronizzare tutta una serie di meccanismi automatici che avvengono in modo transparente all'utente e che sono vitali per il corretto funzionamento del framework.

In ogni meccanismo di che implementa le logiche di *publish/subscribe* per le notifiche di eventi ci sono fondamentalmente due attori:

- chi pubblica eventi
- chi si registra a tali eventi per esserne notificato

Nulla vieta che per un dato evento i due attori possano pure coincidere.

Per esempio al salvataggio da parte dell'utente di modifiche su una scheda di dettaglio classe (gwClassList), viene in automatico pubblicato un evento di notifica (riguardante in questo caso la modifica di uno specifico record) al quale tutti gli altri potenziali widget aperti in quel momento possono rispondere di conseguenza. Tipicamente un eventuale widget lista Classe (gwClassList) aperto in quel momento, reagirà a tale evento ricaricando i dati mostrati (preventivamente controllando che la gwClass del record modificato corrisponda alla propria).

Il fatto che un qualsiasi widget sia o meno sensibile ad un certo evento è insito nella natura stessa del widget, ed è in generale una scelta fatta dagli sviluppatori nel momento stesso della creazione del widget. Senza entrare troppo nel dettaglio si può dire che lo sviluppatore in fase di inizializzazione del widget effettua tutte le registrazioni (*subcriptions*) ad eventi che reputa di interesse per il widget, ed implementa le azioni in risposta a tali eventi.

Il meccanismo di *publish/subscribe* può alla bisogna essere usato dall'utente configuratore per implementare logiche di sincronizzazione fra i vari componenti. Tipicamente si tratta di operazioni di refresh di store e UI. I refresh in genere sono automatici in Geoweb per le operazioni standard, ma talune volte possono dover essere espressamente invocati. Esempi tipici possono essere azioni javascript che causano l'esecuzione di file .groovy lato server, ed il codice contenuto in questi .groovy effettua modifiche su un set di dati, che potenzialmente è usato da altri widget aperti nell'applicazione client in quel momento. Queste azioni possono essere lanciate dai componenti:

- *gwClassList*
- *gwClassDetail*
- *gwTaskDetail*

- *leafItem* di tipo *gwAction*
- *gwHtmlReport*
- o in un qualunque punto del codice, dato che esiste una vasto set di js API richiamabile ovunque

Un esempio specifico è rappresentato dalle azioni javascript che avviano processi, la cui esecuzione causa modifiche su un set di dati condivisi. Tipicamente alla fine di tali processi viene eseguito un .groovy dove spesso si effettuano cambi di stato di uno o più record, che non appartengono alla stessa gwClass di processo. Tali cambi di stato impattano lato client, per esmepio, con la visualizzazione in tutte le gwClassList che, per parzializzare i dati, magari usano filtri statici sul valore del campo dello stato.

In linea di massima l'utente configuratore di Geoweb si trova ad effettuare delle pubblicazioni di eventi. Le sottoscrizioni sono di norma già tutte state fatte dagli sviluppatori.

In generale l'utente configuratore deve utilizzare la funzione publish(..) dell'oggetto topic, eventualmente da richiedere tramite require, cosi:

```
require(
    ['dojo/topic'],
    function(topic){
        ...
    }
);
```

Il primo parametro, obbligatorio, è una string che rappresenta la firma dell'evento, mentre il secondo è opzionale, ed è eventualmente usato dal sottoscrittore dell'evento per scegliere come comportarsi in risposta della notifica. In generale, per gli eventi in cui il sottoscrittore risponde con un refresh , la politica usata da Geoweb è la seguente: se si passano degli specifici valori in params essi vengono usati come eventuale criterio di match restrittivo per effettuare o meno il refresh. Se essi non sono presenti, non vengono valutati ed il refresh avviene comunque.

Gli eventi che vengono notificati/sottoscritti in Geoweb sono molti. Si lascia la descrizione dettaglia degli eventi nelle specifiche schede di ogni componente.

Esempi

Publish di notifica della modifca su record della classe 'jor_odl' effettuato dopo esecuzione codice groovy (che effettua le modifiche server side)

```
var scriptName = 'groovy_name';
var parameterMap = {};
var callback = function(responseHashMap){
    //this callback will be executed client side,
    //one time that the .groovy code execution  will be ended server side

    //notifying all interested client side opened widget
    //about record changes (done server side)
    require(['dojo/topic'], function(topic){
        var gwClassName = 'jor_odl';
        var params = { className: gwClassName};
```

```
        topic.publish('gwClass/update_'+gwClassName, params);
      });
};

//invoking a groovy execution that will make some updates on 'jor_odl'
gwClass records
//configurator user should to know gwClass name
groovyActionGeneric(scriptName, parameterMap, callback);
```

Per la pubblicazione degli eventi più comuni sono state create apposite function di API, che non hanno bisogno di fare esplicitamente il require del modulo dojo/topic (Lo fanno loro al loro interno).

Elenco function API

- gwClass insert
- gwClass update
- gwClass delete
- gwClass insert multiple
- gwClass update multiple
- gwClass delete multiple
- gwMap refresh layer
- gwMap reload

gwClass insert

Pubblicazione evento legato alla creazione di un nuovo record su una gwClass. Ad essere sensibili a questa categoria di publish sono tutti i widget che mostrano i record di classe. Quindi, tipicamente, la lista gwClassList ed i suoi derivati (gwClassPrefilteredList, etc..). Più in generale saranno sensibili tutti i widget nella cui configurazione compare il metadato gwClass (tramite gwClassName) e che ne mostrano o fanno comunque uso dei relativi record.

Esempi

Causa il refresh di tutti i widget sensibili.

```
var gwClassName = 'gw_class_name';
publishGwClassInsert(gwClassName);
```

In una riga (usato nelle report Html):

```
var gwClassName = 'gw_class_name'; publishGwClassInsert(gwClassName);
```

gwClass update

Pubblicazione evento legati alla modifica di un record su una gwClass. Ad essere sensibili a questa

categoria di publish sono tutti i widget che mostrano i record di classe. Quindi, tipicamente, la lista gwClassList ed i suoi derivati (gwClassPrefilteredList, etc..). Più in generale saranno sensibili tutti i widget nella cui configurazione compare il metadato gwClass (tramite gwClassName) e che ne mostrano o fanno comunque uso dei relativi record. Anche la scheda di dettaglio gwClassDetail sarà sensibile (a differenza dell'evento di notifica inserimento di un nuovo record).

Se params è presente, i sottoscrittori che rispondono alla notifica con un refresh, verificheranno il match dell'eventuale paramero itemId. Se params è omesso, effettueranno comunque il refresh.

Esempi

Causa il refresh di tutti i widget sensibili.

```
var gwClassName = 'gw_class_name';
publishGwClassUpdate(gwClassName);
```

In una riga (usato nelle report Html).

```
var gwClassName = 'gw_class_name'; publishGwClassUpdate(gwClassName);
```

Causa il refresh di tutti i widget sensibili, che mostrano il record con lo itemId specificato. (Il refresh sarà per esempio effettuato solo dallo specifico gwClassDetail, se aperto)

```
var gwClassName = 'gw_class_name';
var params = {itemId: '123'};
publishGwClassUpdate(gwClassName, params);
```

In una riga (usato nelle report Html):

```
var gwClassName = 'gw_class_name'; var params = {itemId: '123'};
publishGwClassUpdate(gwClassName, params);
```

gwClass delete

Pubblicazione evento legati alla cancellazione di un record di una gwClass. Ad essere sensibili a questa categoria di publish sono tutti i widget che mostrano i record di classe. Quindi, tipicamente, la lista gwClassList ed i suoi derivati (gwClassPrefilteredList, etc..). Più in generale saranno sensibili tutti i widget nella cui configurazione compare il metadato gwClass (tramite gwClassName) e che ne mostrano o fanno comunque uso dei relativi record. Anche la scheda di dettaglio gwClassDetail sarà sensibile (a differenza dell'evento di notifica inserimento di un nuovo record) e risponderà chiudendosi.

Se params è presente, i sottoscrittori che rispondono alla notifica con un refresh, verificheranno il match dell'eventuale paramero itemId. Se params è omesso, effettueranno comunque il refresh.

Esempi

Causa il refresh di tutti i widget sensibili.

```
var gwClassName = 'gw_class_name';
publishGwClassDelete(gwClassName);
```

In una riga (usato nelle report Html).

```
var gwClassName = 'gw_class_name'; publishGwClassDelete(gwClassName);
```

Causa il refresh di tutti i widget sensibili, che mostrano il record con lo itemId specificato. (Il refresh sarà per esempio effettuato solo dallo specifico gwClassDetail, se aperto)

```
var gwClassName = 'gw_class_name';
var params = {itemId: '123'};
publishGwClassDelete(gwClassName, params);
```

In una riga (usato nelle report Html):

```
var gwClassName = 'gw_class_name'; var params = {itemId: '123'};
publishGwClassDelete(gwClassName, params);
```

gwClass insert multiple

Pubblicazione evento legato alla creazione di un nuovi record, su più gwClass.

Ad essere sensibili a questa categoria di publish sono tutti i widget che mostrano i record di classe. Quindi, tipicamente, la lista gwClassList ed i suoi derivati (gwClassPrefilteredList, etc..). Più in generale saranno sensibili tutti i widget nella cui configurazione compare il metadato gwClass (tramite gwClassName) e che ne mostrano o fanno comunque uso dei relativi record.

Esempi

Causa il refresh di tutti i widget sensibili per le classi nell'array.

```
var gwClassNames = ['gw_class_name_1', 'gw_class_name_2',
'gw_class_name_3'];
publishMultiGwClassInsert(gwClassNames);
```

In una riga (usato nelle report Html):

```
var gwClassNames = ['gw_class_name_1', 'gw_class_name_2',
'gw_class_name_3']; publishMultiGwClassInsert(gwClassNames);
```

gwClass update multiple

Pubblicazione evento legati alla modifica di più record già esistenti, su più gwClass.

Ad essere sensibili a questa categoria di publish sono tutti i widget che mostrano i record di classe. Quindi, tipicamente, la lista gwClassList ed i suoi derivati (gwClassPrefilteredList, etc..). Più in generale saranno sensibili tutti i widget nella cui configurazione compare il metadato gwClass (tramite gwClassName) e che ne mostrano o fanno comunque uso dei relativi record.

Anche la scheda di dettaglio gwClassDetail sarà sensibile (a differenza dell'evento di notifica inserimento di un nuovo record).

Se params è presente, i sottoscrittori che rispondono alla notifica con un refresh, verificheranno il match dell'eventuale paramero itemId. Se params è omesso, effettueranno comunque il refresh.

Esempi

Causa il refresh di tutti i widget sensibili per le classi nell'array.

```
var gwClassNames = ['gw_class_name_1', 'gw_class_name_2',
'gw_class_name_3'];
publishMultiGwClassUpdate(gwClassNames);
```

In una riga (usato nelle report Html).

```
var gwClassNames = ['gw_class_name_1', 'gw_class_name_2',
'gw_class_name_3']; publishMultiGwClassUpdate(gwClassNames);
```

Causa il refresh di tutti i widget sensibili per le classi nell'array, che mostrano almeno un record con chiave valorizzata come uno degli itemId di competenza. Il params di indice n, si riderisce alla classe di indice n, e può essere a sua volta un array di params, il che sta a significare che per la classe di indice n sono avvenute modifiche su piu record. Il refresh sarà per esempio effettuato solo dallo specifico gwClassDetail, se aperto.

```
var gwClassNames = ['gw_class_name_1', 'gw_class_name_2',
'gw_class_name_3'];
var paramsList = [[{itemId: '1a'}, {itemId: '1b'}], {itemId: '2'}, {itemId:
'3'}];
publishMultiGwClassUpdate(gwClassNames, paramsList);
```

In una riga (usato nelle report Html):

```
var gwClassNames = ['gw_class_name_1', 'gw_class_name_2',
'gw_class_name_3']; var paramsList = [[{itemId: '1a'}, {itemId: '1b'}],
{itemId: '2'}, {itemId: '3'}]; publishMultiGwClassUpdate(gwClassNames,
paramsList);
```

gwClass delete multiple

Pubblicazione evento legati alla cancellazione di più record già esistenti, su più gwClass.

Ad essere sensibili a questa categoria di publish sono tutti i widget che mostrano i record di classe.

Quindi, tipicamente, la lista gwClassList ed i suoi derivati (gwClassPrefilteredList, etc..). Più in generale saranno sensibili tutti i widget nella cui configurazione compare il metadato gwClass (tramite gwClassName) e che ne mostrano o fanno comunque uso dei relativi record.

Anche la scheda di dettaglio gwClassDetail sarà sensibile (a differenza dell'evento di notifica inserimento di un nuovo record) e risponderà chiudendosi.

Se params è presente, i sottoscrittori che rispondono alla notifica con un refresh, verificheranno il match dell'eventuale paramero itemId. Se params è omesso, effettueranno comunque il refresh.

Esempi

Causa il refresh di tutti i widget sensibili per le classi nell'array.

```
var gwClassNames = ['gw_class_name_1', 'gw_class_name_2',
'gw_class_name_3'];
publishMultiGwClassDelete(gwClassNames);
```

In una riga (usato nelle report Html).

```
var gwClassNames = ['gw_class_name_1', 'gw_class_name_2',
'gw_class_name_3']; publishMultiGwClassDelete(gwClassNames);
```

Causa il refresh di tutti i widget sensibili per le classi nell'array, che mostrano almeno un record con chiave valorizzata come uno degli itemId di competenza. Il params di indice n, si riderisce alla classe di indice n, e può essere a sua volta un array di params, il che sta a significare che per la classe di indice n sono avvenute cancellazioni su piu record. Si chiuderà solo lo specifico gwClassDetail, se aperto.

```
var gwClassNames = ['gw_class_name_1', 'gw_class_name_2',
'gw_class_name_3'];
var paramsList = [[{itemId: '1a'}, {itemId: '1b'}], {itemId: '2'}, {itemId:
'3'}];
publishMultiGwClassDelete(gwClassNames, paramsList);
```

In una riga (usato nelle report Html):

```
var gwClassNames = ['gw_class_name_1', 'gw_class_name_2',
'gw_class_name_3']; var paramsList = [[{itemId: '1a'}, {itemId: '1b'}],
{itemId: '2'}, {itemId: '3'}]; publishMultiGwClassDelete(gwClassNames,
paramsList);
```

gwMap refresh layer

La function publishGwMapForceMgLayersRefresh() pubblica l'evento che forza, da parte del widget mappa/planimetria, una nuova richiesta al server MapGuide dell'immagine dei layer attivi. Ad essere sensibili a questo publish saranno tutti widget mappa 2D. In generale il cambiamento delle entità raffigurate nei layer Mapguide è gestito in automatico per le operazioni di modifica eseguite

all'interno della scehda mappa/planimetria. Per esempio, l'editing in mappa di una entità si conclude, in maniera automatica, con il refresh di tutti i layer in mappa.

Con params null agisce su tutte le mappe. Se params è passato, viene valutato l'eventuale match del parametro *gwMapName* ed, in seconda battuta (in caso di match di gwMapName) dell'eventuale parametro *planId*.

Esempi

Causa il refresh dei layer di tutti i widget mappa aperti.

```
publishGwMapForceMgLayersRefresh();
```

Causa il refresh dei layer di tutti i widget mappa e planimetria aperti, che sono basati sulla gwMap che nel GeoAdmin ha per nome gwMapName.

```
var params = {gwMapName: 'map_name'};
publishGwMapForceMgLayersRefresh(params);
```

Causa il refresh dei layer di tutti i widget mappa e planimetria aperti, che sono basati sulla gwMap che nel GeoAdmin ha per nome gwMapName e che hanno stesso planId.

```
var params = {gwMapName: 'map_name', planId: 123};
publishGwMapForceMgLayersRefresh(params);
```

gwMap reload

La function publishGwMapReload() pubblica l'evento che forza, da parte del widget mappa/planimetria, il suo ricaricamento totale.

Con params null agisce su tutte le mappe. Se params è passato, viene valutato l'eventuale match del parametro *gwMapName* ed, in seconda battuta (in caso di match di gwMapName) dell'eventuale parametro *planId*.

Esempi

Causa il refresh dei layer di tutti i widget mappa aperti.

```
publishGwMapReload();
```

Causa il refresh dei layer di tutti i widget mappa e planimetria aperti, che sono basati sulla gwMap che nel GeoAdmin ha per nome gwMapName.

```
var params = {gwMapName: 'map_name'};
publishGwMapReload(params);
```

Causa il refresh dei layer di tutti i widget mappa e planimetria aperti, che sono basati sulla gwMap

che nel GeoAdmin ha per nome gwMapName e che hanno stesso planId.

```
var params = {gwMapName: 'map_name', planId: 123};
publishGwMapReload(params);
```

Altri eventi di interesse sottoscrivibili

- gwClassDetail ready
- gwClassDetail formManagerReady
- gwClassDetail gwAttributeReady
- gwClassDetail allFormManagerStarted
- gwClassDetail formManagerStarted
- gwClassDetail onClose
- gwClassDetail gwset value
- gwClassDetail gwset required
- gwClassDetail gwset readonly
- gwClassDetail gwset disabled
- gwClassDetail gwset hidden
- gwClassDetail changed value
- gwClassDetail changed required
- gwClassDetail changed readonly
- gwClassDetail changed disabled
- gwClassDetail changed hidden

gwClassDetail ready

L'evento con firma 'gwClassDetail/ready' è pubblicato quando il dettaglio di classe è nello stato *ready*. Essere nello stato *ready* per il gwClassDetail comporta che:

- Il gwClassDetail espone gia tutto il set delle sue javascript API
- tutti i widget sono stati correttamente inizializzati e dotati del set di javascript API standard trasversali a tutti i gwAttribute widget
- sono stati inizializzati tutti i vincoli definiti dai gwEventHandlers (*Eventi Attributi*)
- sono state già eseguite tutte le eventuali gwAction di *dettaglio pronto*

Più in dettaglio, lo stato *ready* è pubblicato solo quando tutti i formaManger, ognuno contenente tutti gli attributi di un gwAttributesGroup, sono a loro volta *ready*. A loro volta i formaManger sono *ready* solo quando tutti i singoli gwAttribute hanno notificato che sono nello stato *ready*.

Quindi in generale, la mancata pubblicazione di questo evento segnala che c'è un qualche errore di configurazione (o sul codice dei widget in corso di sviluppo), riguardante almeno un gwAttribute presente nel gwClassDetailLayout corrente.

Parametri

- **params**: Object, contenente:
  - **gwClassName**: String, nome della gwClass

- **itemId**: String, identificativo del gwRecord
- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database

Esempi

```
var handle = topic.subscribe('gwClassDetail/ready', function(params){
    if(params && params.gwClassName==gwClassName && params.itemId==itemId &&
params.detailContainerId==detailContainerId){
        handle.remove();
        alert('gwClassDetail is ready!');
    }
});
this.own(handle);
```

gwClassDetail formManagerReady

L'evento con firma 'gwClassDetail/formManagerReady ' è pubblicato quando il singolo formaManger, contentente tutti i widget degli attributi di un singolo gwAttributeGroup è nello stato *ready*. I formaManger sono *ready* solo quando tutti i singoli gwAttribute ivi contenuti hanno notificato che sono nello stato *ready*.

Parametri

- **params**: Object, contenente:
  - **gwClassName**: String, nome della gwClass
  - **itemId**: String, identificativo del gwRecord
  - **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
  - **detailContainerId**: String, id del gwClassDetail
  - **formManager**: Object, riferimento all'istanza del formManager
  - **formManagerId**: String, id del formManager
  - **attributeGroup**: String, nome del gwAttributeGroup

Esempi

```
var handle = topic.subscribe('gwClassDetail/formManagerReady ',
function(params){
    if(params && params.gwClassName==gwClassName && params.itemId==itemId &&
params.detailContainerId==detailContainerId){
        handle.remove();
        alert('formManager with id='+params.formManagerId+' is ready!');
    }
});
```

```
this.own(handle);
```

gwClassDetail gwAttributeReady

L'evento con firma 'gwClassDetail/gwAttributeReady' è pubblicato quando il singolo widget del gwAttribute è nello stato *ready*.

Parametri

- **params**: Object, contenente:
  - **gwClassName**: String, nome della gwClass
  - **itemId**: String, identificativo del gwRecord
  - **detailContainerId**: String, id del gwClassDetail
  - **formManagerId**: String, id del formManager
  - **attributeGwid**: Integer, id del gwAttribute

Esempi

```
var handle = topic.subscribe('gwClassDetail/gwAttributeReady',
function(params){
    if(params && params.gwClassName==gwClassName && params.itemId==itemId &&
params.detailContainerId==detailContainerId){
        handle.remove();
        alert('gwAttributewith id='+params.attributeGwid+' is ready!');
    }
});
this.own(handle);
```

gwClassDetail allFormManagerStarted

L'evento con firma 'gwClassDetail/allFormManagerStarted ' è pubblicato quando è pubblicato solo quando tutti i formaManger, ognuno contenente tutti gli attributi di un gwAttributesGroup, sono a loro volta *started*. Lo stato *started* è differente da quello *ready*. E' stato introdotto, come stato intermedio prima di *ready*, data la necessità da parte di alcuni widget, per inizializzarsi correttamente, di recuperare stati ed in generale informazioni contenuti in altri widget. Queste informazioni sono già disponibili quando loro sono nello stato *started*. Per esempio alcuni widget hanno parametri *queryClause* contenenti espressioni *#{..}* che fanno riferimento a valori contenuti in altri widget e che vanno risolti a *runtime*. Cosi il widget può recuperare la lista dei record da mostare impostando correttamente i filtri derivanti dal queryClause.

Parametri

- **params**: Object, contenente:
  - **gwClassName**: String, nome della gwClass
  - **itemId**: String, identificativo del gwRecord

- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database

Esempi

```
var handle = topic.subscribe('gwClassDetail/allFormManagerStarted ',
function(params){
    if(params && params.gwClassName==gwClassName && params.itemId==itemId &&
params.detailContainerId==detailContainerId){
        handle.remove();
        alert('gwClassDetail allFormManagerStarted !');
    }
});
this.own(handle);
```

gwClassDetail formManagerStarted

L'evento con firma 'gwClassDetail/formManagerStarted ' è pubblicato quando il singolo formaManger, contenente tutti gli attributi di un gwAttributesGroup, sono a loro volta *started*.

Parametri

- **params**: Object, contenente:
  - **gwClassName**: String, nome della gwClass
  - **itemId**: String, identificativo del gwRecord
  - **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
  - **detailContainerId**: String, id del gwClassDetail
  - **formManager**: Object, riferimento all'istanza del formManager
  - **formManagerId**: String, id del formManager
  - **attributeGroup**: String, nome del gwAttributeGroup

Esempi

```
var handle = topic.subscribe('gwClassDetail/formManagerStarted ',
function(params){
    if(params && params.gwClassName==gwClassName && params.itemId==itemId &&
params.detailContainerId==detailContainerId){
        handle.remove();
        alert('formManager with id='+params.formManagerId+' is started!');
    }
});
this.own(handle);
```

gwClassDetail onClose

L'evento con firma 'gwClassDetail/onClose ' è pubblicato quando il singolo il dettaglio di classe viene chiuso programmaticamente o tramite interazione con la UI da parte dell'utente.

Parametri

- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API

Esempi

```
var handle = topic.subscribe('gwClassDetail/onClose',
function(detailContainer){
    if(detailContainer.id==detailContainerId){
        handle.remove();
        alert('detailContainerwith id='+detailContainer.id+' is closed!');
    }
});
this.own(handle);
```

gwClassDetail gwset value

L'evento con firma 'gwClassDetail/gwset/value' è pubblicato quando la proprietà *value* di un gwAttribute widget viene impostato tramite l'apposita javascript API. Tale evento è pubblicato, e gestito in automatico, per il corretto funzionamento dei gwAttribute eventHandlers (*eventi attributi*), ma può essere, ove necessario, liberamente sottoscritto.

Parametri

* **params**: Object, contenente:

- **value**: Object, il nuovo valore della proprietà 'value'
- **itemId**: String, identificativo del gwRecord
- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database
- **gwAttributeName**: String, name del gwAttribute
- **gwAttributeGwid**: String, id del gwAttribute

Esempi

```
var handle = topic.subscribe('gwClassDetail/gwset/value', function(params){
```

```
    if(gwAttributeName.id==params.gwAttributeName){
        handle.remove();
        alert('The property \'value\' for widget of the gwAttribute with
name='+gwAttributeName+' has been set!');
    }
});
this.own(handle);
```

gwClassDetail gwset required

L'evento con firma 'gwClassDetail/gwset/required' è pubblicato quando la proprietà *required* di un gwAttribute widget viene impostato tramite l'apposita javascript API. Tale evento è pubblicato, e gestito in automatico, per il corretto funzionamento dei gwAttribute eventHandlers (*eventi attributi*), ma può essere, ove necessario, liberamente sottoscritto.

Parametri

* **params**: Object, contenente:

- **required**: Object, il nuovo valore della proprietà 'required'
- **itemId**: String, identificativo del gwRecord
- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database
- **gwAttributeName**: String, name del gwAttribute
- **gwAttributeGwid**: String, id del gwAttribute

Esempi

```
var handle = topic.subscribe('gwClassDetail/gwset/required ',
function(params){
    if(gwAttributeName.id==params.gwAttributeName){
        handle.remove();
        alert('The property \'required \' for widget of the gwAttribute with
name='+gwAttributeName+' has been set!');
    }
});
this.own(handle);
```

gwClassDetail gwset readonly

L'evento con firma 'gwClassDetail/gwset/readonly' è pubblicato quando la proprietà *readonly* di un gwAttribute widget viene impostato tramite l'apposita javascript API. Tale evento è pubblicato, e gestito in automatico, per il corretto funzionamento dei gwAttribute eventHandlers (*eventi attributi*), ma può essere, ove necessario, liberamente sottoscritto.

Parametri

* **params**: Object, contenente:

- **readonly**: Object, il nuovo valore della proprietà 'readonly'
- **itemId**: String, identificativo del gwRecord
- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database
- **gwAttributeName**: String, name del gwAttribute
- **gwAttributeGwid**: String, id del gwAttribute

Esempi

```
var handle = topic.subscribe('gwClassDetail/gwset/readonly ',
function(params){
    if(gwAttributeName.id==params.gwAttributeName){
        handle.remove();
        alert('The property \'readonly \' for widget of the gwAttribute with
name='+gwAttributeName+' has been set!');
    }
});
this.own(handle);
```

gwClassDetail gwset disabled

L'evento con firma 'gwClassDetail/gwset/disabled' è pubblicato quando la proprietà *disabled* di un gwAttribute widget viene impostato tramite l'apposita javascript API. Tale evento è pubblicato, e gestito in automatico, per il corretto funzionamento dei gwAttribute eventHandlers (*eventi attributi*), ma può essere, ove necessario, liberamente sottoscritto.

Parametri

* **params**: Object, contenente:

- **disabled**: Object, il nuovo valore della proprietà 'disabled'
- **itemId**: String, identificativo del gwRecord
- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database
- **gwAttributeName**: String, name del gwAttribute
- **gwAttributeGwid**: String, id del gwAttribute

Esempi

```
var handle = topic.subscribe('gwClassDetail/gwset/disabled ',
function(params){
    if(gwAttributeName.id==params.gwAttributeName){
        handle.remove();
        alert('The property \'disabled \' for widget of the gwAttribute with
name='+gwAttributeName+' has been set!');
    }
});
this.own(handle);
```

gwClassDetail gwset hidden

L'evento con firma 'gwClassDetail/gwset/hidden' è pubblicato quando la proprietà *hidden* di un gwAttribute widget viene impostato tramite l'apposita javascript API. Tale evento è pubblicato, e gestito in automatico, per il corretto funzionamento dei gwAttribute eventHandlers (*eventi attributi*), ma può essere, ove necessario, liberamente sottoscritto.

Parametri

* **params**: Object, contenente:

- **hidden**: Object, il nuovo valore della proprietà 'hidden'
- **itemId**: String, identificativo del gwRecord
- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database
- **gwAttributeName**: String, name del gwAttribute
- **gwAttributeGwid**: String, id del gwAttribute

Esempi

```
var handle = topic.subscribe('gwClassDetail/gwset/hidden ',
function(params){
    if(gwAttributeName.id==params.gwAttributeName){
        handle.remove();
        alert('The property \'hidden \' for widget of the gwAttribute with
name='+gwAttributeName+' has been set!');
    }
});
```

gwClassDetail changed value

L'evento con firma 'gwClassDetail/changed/value' è pubblicato quando la proprietà *value* di un gwAttribute widget viene modificata tramite l'apposita javascript API. Tale evento è pubblicato, e gestito in automatico, per il corretto funzionamento dei gwAttribute eventHandlers (*eventi attributi*), ma può essere, ove necessario, liberamente sottoscritto. L'evento è tipicamente pubblicato vincolandolo, tramite il modulo *dojo/after*, all'esecuzione della function *onChange*. Non in tutti i widget l'evento 'gwClassDetail/gwset/value' è eseguito simultaneamente all'evento 'gwClassDetail/changed/value', per questo sono tenuti separati anche se il secondo è sempre successivo al primo.

Parametri

* **params**: Object, contenente:

- **value**: Object, il nuovo valore della proprietà 'value'
- **itemId**: String, identificativo del gwRecord
- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database
- **gwAttributeName**: String, name del gwAttribute
- **gwAttributeGwid**: String, id del gwAttribute

Esempi

```
var handle = topic.subscribe('gwClassDetail/changed/value',
function(params){
    if(gwAttributeName.id==params.gwAttributeName){
        handle.remove();
        alert('The property \'value\' for widget of the gwAttribute with
name='+gwAttributeName+' has changed!');
    }
});
this.own(handle);
```

gwClassDetail changed required

L'evento con firma 'gwClassDetail/changed/required' è pubblicato quando la proprietà *required* di un gwAttribute widget viene impostato tramite l'apposita javascript API. Tale evento è pubblicato, e gestito in automatico, per il corretto funzionamento dei gwAttribute eventHandlers (*eventi attributi*), ma può essere, ove necessario, liberamente sottoscritto.

Parametri

* **params**: Object, contenente:

- **required** : Object, il nuovo valore della proprietà 'required'

- **itemId**: String, identificativo del gwRecord
- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database
- **gwAttributeName**: String, name del gwAttribute
- **gwAttributeGwid**: String, id del gwAttribute

Esempi

```
var handle = topic.subscribe('gwClassDetail/changed/required ',
function(params){
    if(gwAttributeName.id==params.gwAttributeName){
        handle.remove();
        alert('The property \'required \' for widget of the gwAttribute with
name='+gwAttributeName+' has changed!');
    }
});
this.own(handle);
```

gwClassDetail gwset readonly

L'evento con firma 'gwClassDetail/gwset/readonly' è pubblicato quando la proprietà *readonly* di un gwAttribute widget viene impostato tramite l'apposita javascript API. Tale evento è pubblicato, e gestito in automatico, per il corretto funzionamento dei gwAttribute eventHandlers (*eventi attributi*), ma può essere, ove necessario, liberamente sottoscritto.

Parametri

* **params**: Object, contenente:

- **readonly**: Object, il nuovo valore della proprietà 'readonly'
- **itemId**: String, identificativo del gwRecord
- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database
- **gwAttributeName**: String, name del gwAttribute
- **gwAttributeGwid**: String, id del gwAttribute

Esempi

```
var handle = topic.subscribe('gwClassDetail/gwset/readonly ',
function(params){
    if(gwAttributeName.id==params.gwAttributeName){
        handle.remove();
```

```
        alert('The property \'readonly \' for widget of the gwAttribute with
name='+gwAttributeName+' has been set!');
    }
});
this.own(handle);
```

gwClassDetail changed disabled

L'evento con firma 'gwClassDetail/changed /disabled' è pubblicato quando la proprietà *disabled* di un gwAttribute widget viene impostato tramite l'apposita javascript API. Tale evento è pubblicato, e gestito in automatico, per il corretto funzionamento dei gwAttribute eventHandlers (*eventi attributi*), ma può essere, ove necessario, liberamente sottoscritto.

Parametri

* **params**: Object, contenente:

- **disabled**: Object, il nuovo valore della proprietà 'disabled'
- **itemId**: String, identificativo del gwRecord
- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database
- **gwAttributeName**: String, name del gwAttribute
- **gwAttributeGwid**: String, id del gwAttribute

Esempi

```
var handle = topic.subscribe('gwClassDetail/changed /disabled ',
function(params){
    if(gwAttributeName.id==params.gwAttributeName){
        handle.remove();
        alert('The property \'disabled \' for widget of the gwAttribute with
name='+gwAttributeName+' has changed!');
    }
});
this.own(handle);
```

gwClassDetail changed hidden

L'evento con firma 'gwClassDetail/changed/hidden' è pubblicato quando la proprietà *hidden* di un gwAttribute widget viene impostato tramite l'apposita javascript API. Tale evento è pubblicato, e gestito in automatico, per il corretto funzionamento dei gwAttribute eventHandlers (*eventi attributi*), ma può essere, ove necessario, liberamente sottoscritto.

Parametri

* **params**: Object, contenente:

- **hidden**: Object, il nuovo valore della proprietà 'hidden'
- **itemId**: String, identificativo del gwRecord
- **detailContainer**: Object, riferimento all'istanza del gwClassDetail, che espone tutte le relative API
- **detailContainerId**: String, id del gwClassDetail
- **data**: Object, contiene il record relativo al gwClassDetail, cosi come arriva dal database
- **gwAttributeName**: String, name del gwAttribute
- **gwAttributeGwid**: String, id del gwAttribute

Esempi

```
var handle = topic.subscribe('gwClassDetail/changed /hidden ',
function(params){
    if(gwAttributeName.id==params.gwAttributeName){
        handle.remove();
        alert('The property \'hidden \' for widget of the gwAttribute with
name='+gwAttributeName+' has changed!');
    }
});
```

Trick sottoscrizione eventi

Refresh su altra gwClass su modifiche linkListNam

Quando si vuole il refresh automatico di gwClassList/gwClassDetail su classi diverse da quelle coinvolte nell'associazione/disassociazione effettuata da una linkListNam La linkListNam di suo coinvolge 3 classi: start, associazione, target.

dalla 4.7.2 Usare js api **publishGwClassUpdateWhenLinkListNamChanges()**

I parametri llnamStartClassName, llnamAssociationClassName, llnamTargetClassName sono tutti opzionali e possono essere usati come criteri di restrizione per effettuare il publish

Parameters:

- params, Object, required
    - gwClassName, string, required, name of the gwClass to update
    - llnamStartClassName, string, optional, name of the linkListNam widget start gwClass
    - llnamAssociationClassName, string, optional, name of the linkListNam widget association gwClass
    - llnamTargetClassName, string, optional, name of the linkListNam widget end gwClass

```
var params = {
        gwClassName: 'aaa',
```

```
        llnamStartClassName: 'bbb',
        llnamAssociationClassName: 'ccc',
        llnamTargetClassName: 'ddd'
    }
    var handle = publishGwClassUpdateWhenLinkListNamChanges(params);
    //... (later on code)
    handle.remove();
```

pre 4.7.2

Nel codice sotto le liste e i dettagli della classe gwClassNameTarget, vengono aggiornate su tutte le associazioni fatte da una linkListNam che hanno come target gwClassNameLinkListNamTarget (unico criterio di restrizione)

```
//automatically refreshing gwClassNameTarget gwClassList and gwClassDetail
//when any modification is performed on a LinkListNam widget
//var gwClassNameSource = 'gwa_role_r_func'; //not used
var gwClassNameTarget = 'gwa_role_filter_conf';
var gwClassNameLinkListNamTarget = 'gwa_func';
require(['dojo/topic'], function(topic){
    topic.subscribe('linkListNam/association/ended', function(data){
        //gwGrid is the LinkListNam widget
        //gwGrid.gwClassName is the target gwClass
        if(data.gwGrid.gwClassName===gwClassNameLinkListNamTarget){
            var params = null; //it force to reload all related gwClassList
and gwClassDetail
            publishGwClassUpdate(gwClassNameTarget, params);
        }
    });
});
```

From:
https://wiki.geowebframework.com/ - **GeowebFramework**

Permanent link:
https://wiki.geowebframework.com/doku.php?id=custom:api_js_publish_event

Last update: **2024/09/05 17:36**