

Superclasse

- **cwbBpaWizard**: Finestra di gestione wizard da estendere

Interfacce

- **wizardable**: Interfaccia da implementare obbligatoriamente

Wizard Container

Per creare un wizard va creata una form 'container' con all'interno:

- * **la buttonBar (tasti gestiti: Avanti, Indietro, Annulla e Concludi).**
- * **un div segnaposto chiamato 'divGestione' su cui verranno inserite le form dei vari step.**

Questa form container dovrà estendere la classe cwbBpaWizard mentre le form dei singoli step dovranno implementare l'interfaccia wizardable. Sulla form container nel metodo initVars vanno valorizzate le seguenti proprietà:

- * **nameForm**: il nome della form container
- * **firstStepName**: Il nome della form utilizzata come primo step
- * **lastStepName**: Il nome della form utilizzata come ultimo step
- * **DBName**: Il nome del db da usare per la connessione (es. 'CITYWARE')

Il primo e l'ultimo step sono quindi obbligatoriamente fissi, mentre tutte le form intermedie sono dinamiche.

Nella superclasse è presente una variabile 'navigationRules' che è una linkedlist che gestisce il flusso. La variabile navigationRules ha un metodo current (\$this->navigationRules->current()) che specifica la form corrente in cui ci troviamo e una variabile currentKey (\$this->navigationRules->currentKey()) che indica il numero di passo.

Nel metodo preNext va gestito il comportamento del wizard al click del tasto avanti (va impostato qual'è lo step successivo e le azioni/parametri da passare). Come parametro arrivano 'currentStep' e 'currentKey' che contengono il nome della form dello step corrente e il numero di step. Per aggiungere uno step alla navigationRules va chiamato il metodo addStepToNavigationRules passandogli come parametro il nome della form successiva

(\$this->addStepToNavigationRules('nomeDelloStepSuccessivo')).

es implementazione preNext:

```
protected function preNext($currentStep, $currentKey) {
    switch ($currentStep) {
        case 'cwdForm1':
            $x = $_POST['cwdForm1_x'];
            switch ($x) {
                case "1":
                    $this->addStepToNavigationRules('cwdForm2'); // se x=1
aggiungo cwdForm2 come step successivo
                    // codice....
                    break;
                case 2:
                    $this->addStepToNavigationRules('cwdForm3'); // se x=2
aggiungo cwdForm3 come step successivo
                    // codice....
                    break;
            }
            break;
        case 'cwdForm2':
            // se non aggiungo uno step successivo, vado allo step finale
            // codice....
            break;
    }
}
```

Implementando l'interfaccia wizardable, verrà richiesto di implementare i metodi:

* **validaWizardStep(\$formData, &\$msg)** in cui va gestita la validazione della form al click del tasto 'avanti'. Il metodo deve tornare true o false in base al risultato e concatenare i messaggi di errore su \$msg.

* **setValueToSave()** che serve per settare eventuali var non presenti in \$_POST che si vogliono mantenere in cache. Deve tornare un array di elementi con chiave il nome del setter delle proprietà da gestire e valore i rispettivi record contenuti. es. Si deve utilizzare per resettare il valore delle grid. Questo perché dalla \$_POST non è possibile reperire il contenuto della grid per metterlo in cache e quindi all'indietro si perderebbe il contenuto.

* **getAppliedCache()/setAppliedCache(\$appliedCache);** getter e setter da implementare (creare anche la property private \$appliedCache; e gestirla nel getter/setter) per capire quando la form è stata aperta applicando la cache (getAppliedCache = true) oppure no. Se è stata applicata la cache ad esempio le combo devono essere inizializzate col valore precedente invece che con 'Selezionare..' di default.

All'evento 'indietro' la cache è automatica, mentre per gestire la cache all'avanti va implementato il metodo **applyCacheNext(\$currentNameForm, \$oldNameForm, \$cacheValue)**. Questo metodo torna la form corrente in cui si sta andando, quella precedente e la cache della form corrente. I valori

in cache vanno confrontati con quelli in \$_POST per capire se sono state eseguite modifiche alla form precedente che annullano la validità della cache sulla form successiva (return false) oppure no (return true). Se questo metodo torna true verrà applicata la cache sennò no.

Per gestire il salvataggio finale/azioni varie è possibile in ogni step, aggiungere delle operazioni o salvarsi dei dati.

* **Aggiungere Dati:** `$this->addFixedParameterCache($key, $value, $formName = null, $formKey = null);`

In questo modo si aggiunge un valore in cache per poi riutilizzarlo alla fine (metodo postComplete). Se viene passata \$formName oppure \$formName e \$formKey questo valore viene pulito in automatico facendo indietro dalla form '\$formName' se invece non si passa \$formName il valore rimane fisso ed al limite può essere cancellato a mano (cleanFixedParameterCache). es.

```
$this->addFixedParameterCache('CODPROF', $_POST['cwdDtaTitstuWiz_CODPROF'],  
$currentStep, $currentKey);
```

* **Aggiunta Operazioni:** `$this->addOperationCache($formDataName, $formDataKey, $operationKey, $operation, $table, $value, $recordInfo = null)`

In questo modo si aggiungo operazioni che vengono eseguite in maniera automatica al click finale su 'Conferma'.

es.

```
$value = array('PROGENTE' => 1, 'DESCRIZ' => "Prova", 'ALIAS' => "PROVA");  
$this->addOperationCache($currentStep, $currentKey, "salvoGruppi",  
itaModelService::OPERATION_INSERT, "BOR_GRUPPI", $value);
```

Al click su 'Conferma', viene aperta la transazione, vengono eseguite in automatico tutte le operazioni e poi viene chiamato il postComplete(\$db) in cui si possono eseguire altre operazioni (anche basandosi sui dati 'appoggiati' in cache tramite addFixedParameterCache). Alla fine di tutto viene chiusa la transazione. Al postComplete va usato come \$DB quello che arriva come parametro in modo da mantenere la transazione. Quindi tutte le operazioni eseguite al 'Conferma' sono nella stessa transazione e se fallisce un'operazione, falliscono tutte. Se vanno eseguite delle operazioni fuori transazione, al 'conferma' è disponibile anche il metodo generateOutput() che parte dopo postComplete ed è fuori transazione.

Per gestire i messaggi finali, in caso di errore o esito positivo, nella superclasse ci sono \$completeErrorMsg e \$completeMsg che vanno valorizzati con i messaggi da stampare (\$completeMsg se non valorizzato di default prende: 'Pratica conclusa con successo').

From:

<https://wiki.nuvolaitalsoft.it/> - **wiki**

Permanent link:

https://wiki.nuvolaitalsoft.it/doku.php?id=sviluppo:cityware_wizard

Last update: **2018/03/19 10:45**

