

Web API
Guida all'uso



Prima edizione – marzo 2014

Sommario

Web API.....	4
1 Introduzione.....	4
2 Come creare Web API con Instant Developer.....	4
3 Specifiche delle chiamate a Web API	5
4 Esempi di chiamate	7
5 Algoritmo della gestione di una chiamata	12
6 Esempi di codice con personalizzazioni	16
7 Come testare una Web API	21

Web API

1 Introduzione

Una delle modalità più diffuse per la commercializzazione di prodotti software è il [SaaS](#) (Software as a Service) che consente una completa fruibilità via web e l'installazione in ambiente cloud.

La fruibilità via web è resa possibile dalle [Web API](#) (Application Programming Interface) ovvero interfacce esposte sul web che permettono di scambiare dati tra un qualsiasi client e un'applicazione remota.

Il modello architetturale comunemente usato nelle Web API è il modello [RESTful](#). I principi fondamentali su cui si basa sono i seguenti:

- lo stato dell'applicazione e le funzionalità sono divisi in Risorse WEB;
- ogni risorsa è unica e indirizzabile usando sintassi universale per i link ipertestuali;
- tutte le risorse sono condivise come interfaccia uniforme per il trasferimento di stato tra client e risorse. Ciò consiste in:
 - un insieme vincolato di operazioni ben definite;
 - un insieme vincolato di contenuti, opzionalmente supportato da codice on demand;
 - un protocollo che è Client-Server, Stateless, Cachabile e a livelli.

2 Come creare Web API con Instant Developer

Creare Web API con Instant Developer è molto semplice. Per ogni risorsa che si desidera esporre è infatti sufficiente creare una classe DO con le proprietà necessarie e attivare il servizio *WebAPI*. Si ottengono così automaticamente le seguenti funzionalità:

- lettura, inserimento, eliminazione e aggiornamento di una risorsa dato il suo identificativo;
- ricerca di una collezione di risorse dati i criteri di filtro sulle proprietà.

Inoltre si possono esporre funzionalità specifiche creando metodi nella classe e attivando il flag *WebAPI*.

Ad esempio, si supponga di voler creare una Web API per consultare i prodotti identificati da un codice alfanumerico (quale P10). Innanzitutto si creerà la classe Prodotto trascinando la corrispondente tabella del database sull'applicazione. Attivando il servizio *WebAPI* e compilando l'applicazione sarà già possibile ottenere le

informazioni del prodotto con codice P10 semplicemente digitando nel browser l'URL *http://miodominio/miaapp/Prodotto/P10*.

3 Specifiche delle chiamate a Web API

Ora vediamo nel dettaglio le caratteristiche delle chiamate a Web API create con Instant Developer.

3.1 Chiamate stateless

Le chiamate sono [stateless](#), ovvero ogni richiesta non ha memoria delle precedenti e pertanto non viene gestita la sessione.

Se si desidera una modalità di gestione non stateless, è possibile identificare le chiamate con un token e memorizzare le informazioni di sessione in un database o in una qualsiasi altra fonte dati.

3.2 Formato dei dati

Il formato predefinito dei dati scambiati è JSON, ma è possibile esplicitare il formato desiderato (JSON o XML) indicandolo nell'URL.

Se si desidera ad esempio ottenere il prodotto P10 in formato XML, occorre usare l'URL *http://miodominio/miaapp/Prodotto.xml/P10*.

3.3 Composizione degli URL

L'URL delle chiamate a Web API è composto dalle parti descritte di seguito:

- percorso dell'applicazione (ad esempio *http://miodominio/miaapp*);
- nome della classe ([Tag](#)) comprensivo dell'eventuale namespace o package del componente. Se ad esempio la classe Prodotto è definita in un componente con namespace *com.progamma.catalogo*, l'URL dovrà essere seguito da */com/progamma/catalogo/Prodotto*;
- eventuale formato desiderato; per esplicitare il formato occorre aggiungere al nome della classe la stringa *.json* o *.xml*;
- valori della chiave primaria (PK) separati da barre per le richieste di lettura o eliminazione (ad esempio */P10/2010*);

- criteri di ricerca per le richieste di ricerca o valori delle proprietà per le richieste di aggiornamento (ad esempio `?Year=2010&Supplier=2`); le proprietà sono identificate dal corrispondente valore di [Tag](#);
- valori dei parametri per richieste di metodi custom.

3.4 Tipologie di chiamate

Le tipologie di chiamate che si possono effettuare sono identificate dal metodo HTTP utilizzato come riportato nella tabella seguente:

Metodo HTTP	Tipologia chiamata
GET	Lettura di una risorsa con i valori della PK nell'URL.
GET	Ricerca di risorse con i criteri di ricerca nell'URL.
PUT	Aggiornamento di una risorsa esistente identificata dai valori della PK nell'URL, e dai valori modificati nell'URL oppure nel contenuto.
POST	Inserimento di una nuova risorsa con la serializzazione della risorsa nel contenuto.
DELETE	Eliminazione di una risorsa esistente identificata dai valori della PK nell'URL.
NOMEMETHOD O	Chiamata ad un metodo custom con i parametri nell'URL o nel contenuto.

3.5 Profondità della gerarchia di figli

Quando si effettuano chiamate a funzionalità predefinite del servizio di Web API (GET, PUT, POST e DELETE) è possibile specificare il livello di profondità dei figli che si desidera coinvolgere impostando nella richiesta un header con nome *child-level*.

Se non specificato, il valore predefinito è 0 per GET con criteri di ricerca e 9999 in tutti gli altri casi.

3.6 Metodi custom e limitazioni delle chiamate

Nelle chiamate a metodi custom il nome del metodo da chiamare deve essere specificato nel metodo HTTP della richiesta. È possibile tuttavia che non sia ammesso effettuare la chiamata con un metodo HTTP non standard per diversi motivi:

- il linguaggio usato lato client per effettuare la chiamata non lo permette;

- il firewall del client o del server non lo permette;
- il server web non lo permette.

Qualora si presenti uno di questi casi, occorre usare POST come metodo HTTP e specificare il nome del metodo che si intende chiamare in un header della richiesta con nome *X-HTTP-Override-Method*. Il server controlla se questo header è presente prima di decidere quale operazione effettuare.

3.7 Codifica dei valori

I valori scambiati nelle chiamate a Web API rispettano le codifiche riportate nella seguente tabella:

Tipo di dato	Codifica
Null	I valori null vengono codificati con la stringa <i>~NULL~</i> per distinguerli dal valore stringa vuota
Datetime	Per le date viene utilizzato il formato <i>yyyy-mm-dd hh:nn.ss</i>
Numero con separatore decimale	Per i valori numerici con separatore viene utilizzato sempre il punto come separatore dei decimali e viene omesso il separatore delle migliaia
Boolean	I valori booleani vengono rappresentati con -1 e 0
BLOB	I blob vengono rappresentati nel formato web (ad esempio <i>data:image/gif;base64,...</i>)
DocID	I DocID vengono codificati sotto forma di GUID
IDDocumnt e IDCollection	Solo istanze di queste classi vengono serializzate in XML o JSON; l'uso di un qualsiasi altro tipo di oggetto viene segnalato da Instant Developer in fase di validazione come non utilizzabile in chiamate a Web API

Le proprietà in XML, in JSON e nei criteri di filtro sono identificate dal corrispondente valore di [Tag](#). Queste codifiche devono essere rispettate dai client per i dati inviati e sono rispettate dal server per i dati restituiti.

4 Esempi di chiamate

In questo paragrafo vengono illustrati alcuni esempi di chiamate, uno per ogni tipologia. Negli esempi si suppone che l'applicazione che viene chiamata risponda all'URL <http://www.progamma.com/NewWebApp>.

4.1 Esempio di lettura di risorse

Per ottenere una risorsa identificata dai valori chiave (ad esempio una Fattura con Anno e Numero come PK) occorre effettuare una chiamata di questo tipo:

```
GET http://www.progamma.com/NewWebApp/Fattura/2013/1 HTTP/1.1
```

La risposta potrebbe essere questa:

```
HTTP/1.1 200 OK
Content-type: application/json

{
  Anno : 2013,
  Numero : 1,
  ...
  Riga : [ { id: 1, ... }, { id: 2, ... }, ... ]
}
```

Si può notare che sono state restituite anche le righe della fattura, in quanto è stato omesso l'header child-level (quindi considerato 9999); inoltre il formato usato nella risposta è JSON in quanto non specificato nell'URL.

4.2 Esempio di ricerca di risorse

Per ottenere le risorse che corrispondono ad alcuni criteri di ricerca (ad esempio le fatture dell'anno 2013 e 2014 e in stato C) occorre effettuare una chiamata di questo tipo:

```
GET http://www.progamma.com/NewWebApp/Fattura.xml?Anno=2013;2014&Stato=C
HTTP/1.1
```

La risposta potrebbe essere questa:

```
HTTP/1.1 200 OK
Content-type: application/xml; charset=utf-8

<IDCollection>
  <Fattura Anno="2013" Numero="1" ... />
  <Fattura Anno="2013" Numero="2" ... />
  <Fattura Anno="2013" Numero="3" ... />
</IDCollection>
```


Si può notare che i criteri di ricerca sono specificati nella query string dell'URL utilizzando anche la sintassi QBE (vedi 2013;2014). In questo caso è stato specificato il formato XML nell'URL, pertanto la risposta è nel formato XML. Avendo omesso anche in questo caso l'header *child-level*, esso viene considerato 0 e pertanto non vengono restituite le righe delle fatture.

4.3 Esempio di inserimento di risorse

Per inserire una nuova risorsa (ad esempio una fattura) occorre effettuare una chiamata con i dati della risorsa in formato JSON o XML nel contenuto:

```
POST http://www.progamma.com/NewWebApp/Fattura HTTP/1.1
Content-type: application/json

{
  Anno : 2013,
  Numero : 1,
  ...
  Riga : [ { id: 1, ... }, { id: 2, ... }, ... ]
}
```

Qualora l'inserimento riesca, la risposta sarà di questo tipo:

```
HTTP/1.1 204 OK
```

Se la classe in questione ha un campo contatore come chiave primaria, nella risposta verrà riportato il valore attribuito dal contatore:

```
HTTP/1.1 200 OK
Content-type: text/plain; charset=utf-8
10231
```

Mentre sarà di questo tipo in caso non riesca:

```
HTTP/1.1 500 Internal Server Error
Content-type: text/plain; charset=utf-8
<Messaggi di errore>
```

4.4 Esempio di eliminazione di risorse

Per eliminare una risorsa (ad esempio la fattura dell'anno 2013 con numero 1), occorre effettuare una chiamata di questo tipo:

```
DELETE http://www.progamma.com/NewWebApp/Fattura/2013/1 HTTP/1.1
```

La risposta in caso di esito positivo sarà come quella del caso di inserimento. Se invece la fattura non viene trovata, la risposta sarà di questo tipo:

```
HTTP/1.1 404 Not found
Content-type: text/plain; charset=utf-8

Fattura not found
```

4.5 Esempio di aggiornamento di risorse

Per aggiornare una risorsa (ad esempio la fattura numero 1 dell'anno 2013) si può procedere in due modi:

- passare i valori da modificare nell'URL; in questo caso la chiamata dovrà essere di questo tipo:

```
PUT http://www.progamma.com/NewWebApp/Fattura/2013/1?Stato=C&...
HTTP/1.1
```

- passare i dati da modificare nella fattura e i valori della PK in formato JSON o XML nel contenuto qualora si desideri aggiornare anche i figli (oppure semplicemente se si desidera procedere in questo modo):

```
PUT http://www.progamma.com/NewWebApp/Fattura HTTP/1.1
Content-type: application/json

{
  Anno : 2013,
  Numero : 1,
  Stato : "C"
  ...
  Riga : [ { id: 1, ... }, { id: 2, ... }, ... ]
}
```

La risposta sarà come quella dei casi precedenti.

4.6 Esempio di chiamata a metodo statico

Per chiamare un metodo statico, occorre effettuare una chiamata di questo tipo:

- qualora il metodo sia senza parametri o abbia parametri semplici (ad esempio il conteggio delle fatture di un anno) i valori dei parametri devono essere passati nell'URL:

```
CALCOLA http://www.progamma.com/NewWebApp/Fattura?Anno=2013 HTTP/1.1
```

- qualora invece il metodo abbia almeno un parametro di tipo oggetto (ad esempio IDCollection o IDDocument) i valori dei parametri devono essere passati nel contenuto:

```
POST http://www.progamma.com/NewWebApp/Fattura HTTP/1.1
X-HTTP-Override-Method: CALCOLA
Content-type: application/json

{
  Anno : 2013,
  Numero : 1,
  Stato : "C"
  ...
}
```

La risposta, qualora il metodo restituisca un valore semplice, sarà di questo tipo:

```
HTTP/1.1 200 OK
Content-type: text/plain; charset=utf-8

43
```

Mentre sarà del tipo indicato di seguito qualora il valore di ritorno sia un oggetto:

```
HTTP/1.1 200 OK
Content-type: application/json

{
  Anno : 2013,
  Numero : 1,
  Stato : "C"
  ...
}
```

Nel secondo tipo di chiamata si può notare che il metodo viene specificato nell'header *X-HTTP-Override-Method*.

4.7 Esempio di chiamata a metodo non statico

Per chiamare un metodo non statico si presentano i seguenti casi:

- risorsa identificata tramite PK nell'URL ed eventuali parametri semplici passati via URL:

```
CALCOLASALDO
http://www.progamma.com/NewWebApp/Fattura/2013/1?Sconto=0.5 HTTP/1.1
```

- risorsa identificata tramite PK nell'URL e parametri di tipo oggetto passati nel contenuto:

```
CALCOLASALDO      http://www.progamma.com/NewWebApp/Fattura/2013/1
HTTP/1.1
Content-type: application/json

{
  ScontoObj : { ... }
  ...
}
```

- istanza della risorsa e parametri di tipo oggetto passati nel contenuto; in questo caso l'istanza deve essere identificata dal nome `_ID_INSTANCE`:

```
CALCOLASALDO http://www.progamma.com/NewWebApp/Fattura HTTP/1.1
Content-type: application/json

{
  _ID_INSTANCE:
  {
    Anno : 2013,
    Numero : 1,
    Stato : "C"
    ...
  },
  ScontoObj : { ... }
  ...
}
```

La risposta sarà come quella per i metodi statici.

5 Algoritmo della gestione di una chiamata

In questo paragrafo verrà esaminato nel dettaglio come viene gestita una chiamata a Web API. Innanzitutto è opportuno notare che attivando il servizio WebAPI su una

classe nel file di configurazione dell'applicazione (*web.config* in C#, *web.xml* in Java) vengono aggiunti i mapping necessari per indicare al server web che l'applicazione gestisce anche gli URL specifici delle Web API.

Quando arriva una chiamata al server, il servizio controlla se si tratta di una chiamata a Web API; lo è se l'URL non termina con il documento predefinito (NomeApp.aspx o NomeApp.htm). Se non lo è, la richiesta verrà gestita dal framework RD3. In ogni caso viene sempre notificato l'evento [Initialize](#), nel quale è già possibile testare la funzione *WebApiService.IsWebApiRequest* per determinare il tipo di richiesta.

In seguito, vengono estratti dall'URL il nome della classe, l'eventuale formato e gli eventuali valori della PK. Se non è possibile creare un'istanza della classe, la chiamata termina con il seguente errore:

```
HTTP/1.1 405 Method Not Allowed
Content-type: text/plain; charset=utf-8

Class not found for uri '<path dell'URL>'
```

Viene controllato in seguito che per la classe sia attivato il servizio WebAPI. Nel caso non sia attivato, la chiamata termina con il seguente errore:

```
HTTP/1.1 405 Method Not Allowed
Content-type: text/plain; charset=utf-8

Class '<Tag della classe>' not enabled for WebApi
```

Viene quindi individuato il metodo chiamato tramite verifica della presenza dell'header *X-HTTP-Method-Override*. Per determinare quale metodo è stato chiamato, occorre chiamare la funzione *WebApiService.GetMethod*. Se non si tratta di un metodo base (GET, PUT, POST o DELETE) viene controllato che per il metodo sia attivato il flag WebAPI. Nel caso non sia attivato, la chiamata termina con il seguente errore:

```
HTTP/1.1 405 Method Not Allowed
Content-type: text/plain; charset=utf-8

Method '<nome del metodo>' of class '<Tag della classe>' not enabled for
WebApi
```

Se la classe non ha un metodo con tale nome, la chiamata termina con l'errore:

```
HTTP/1.1 400 Bad Request
Content-type: text/plain; charset=utf-8

Method '<nome del metodo>' of class '<Tag della classe>' not found
```

Se si tratta di un metodo custom, viene controllato che il metodo restituisca un valore serializzabile, ovvero un tipo semplice o un oggetto `IDDocument` o `IDCollection`. In caso contrario, la chiamata termina con il seguente errore:

```
HTTP/1.1 400 Bad Request
Content-type: text/plain; charset=utf-8

Method '<nome del metodo>' of class '<Tag della classe>' has a return
value not serializable
```

Inoltre, viene effettuato lo stesso controllo anche sui parametri. Se il controllo non ha esito positivo, la chiamata termina con il seguente errore:

```
HTTP/1.1 400 Bad Request
Content-type: text/plain; charset=utf-8

Method '<nome del metodo>' of class '<Tag della classe>' has at least a
paramater not serializable
```

Vengono quindi effettuati alcuni controlli di conformità della richiesta, ad esempio:

- parametri forniti quando non necessari o mancanti quando necessari;
- parametri forniti nell'URL in presenza anche del contenuto;
- valori della PK forniti quando non necessari o mancanti quando necessari;
- contenuto fornito quando non necessario o mancante quando necessario.

In tutti questi casi il codice di stato della risposta è sempre 400 Bad Request.

Successivamente viene letto il contenuto, se presente. In caso di errore durante il parsing, la chiamata termina con il seguente errore:

```
HTTP/1.1 400 Bad Request
Content-type: text/plain; charset=utf-8

Unable to parse content of the request. Invalid <formato> format:
<contenuto>
```

Dal contenuto vengono estratti eventuali parametri e/o l'istanza della risorsa. I parametri vengono convertiti nel tipo corretto e l'istanza viene caricata dal formato corretto. In caso di errore, la chiamata termina con codice di stato 400 Bad Request e il testo dell'errore.

Viene in seguito inizializzata la proprietà `WebApiService.ChildLevel` a 0 per le ricerche di risorse o a 9999 in tutti gli altri casi. Se nella richiesta è presente l'header con nome `child-level` la proprietà viene valorizzata con il valore dell'header.

In seguito, se presenti i valori della PK, viene controllato che il loro numero sia corrispondente al numero di PK della classe. In caso contrario, la chiamata termina con il seguente errore:

```
HTTP/1.1 400 Bad Request
Content-type: text/plain; charset=utf-8

Primary key mismatch: the class '<Tag della classe>' has <numero di PK della classe> properties in the primary key, but the values passed are <numero di valori passati>
```

I valori vengono convertiti nei tipi corrispondenti, come per i parametri, e qualora qualche valore non sia convertibile, la chiamata termina con il seguente errore:

```
HTTP/1.1 400 Bad Request
Content-type: text/plain; charset=utf-8
```

Value '<i-esimo valore>' cannot be converted for property '<Tag dell'i-esima proprietà>' of the class '<Tag della classe>'

A questo punto, se non si tratta di un caricamento o ricerca di risorsa (GET), viene chiamato il metodo [LoadFromDB](#) sull'istanza. Qualora il caricamento non sia riuscito, la chiamata termina con il seguente errore:

```
HTTP/1.1 404 Not Found
Content-type: text/plain; charset=utf-8

<DNA del document> not found
```

In seguito, in caso di metodo statico, vengono controllati e convertiti i parametri. In caso di errore la chiamata termina con codice di stato 400 Bad Request e il testo dell'errore.

In caso di ricerca di risorse vengono letti i criteri di ricerca. Qualora non venga trovata una proprietà corrispondente, la chiamata termina con il seguente errore:

```
HTTP/1.1 404 Not Found
Content-type: text/plain; charset=utf-8

Property '<nome del criterio>' not found"
```

In caso di aggiornamento con i valori da modificare nell'URL, vengono copiati i valori nell'istanza. La chiamata termina con codice di stato 400 Bad Request se non viene trovata una proprietà corrispondente oppure se non fallisce la conversione del valore.

In caso di aggiornamento con istanza nel contenuto, viene invece caricata l'istanza originale e, dall'istanza fornita, vengono copiati i valori nelle proprietà corrispondenti. Qualora l'istanza fornita abbia anche dei figli, viene applicato lo stesso procedimento. Qualora un figlio non venga trovato nell'istanza originale, questo viene aggiunto e impostato come inserito. Non viene controllato se nell'istanza fornita mancano figli per un'eventuale cancellazione.

In caso di inserimento di risorsa, semplicemente vengono marcati come inseriti i documenti della gerarchia fino al livello corrispondente alla proprietà *WebApiService.ChildLevel*.

A questo punto vengono copiati gli header della richiesta, che sono interrogabili tramite il metodo *WebApiService.GetHeaders*.

Infine, viene notificato l'evento *OnWebApi* sull'istanza di documento. In tale evento è possibile intervenire sulle operazioni effettuate dal framework fino a quel momento (vedi paragrafo seguente). Qualora l'operazione non sia stata annullata impostando il parametro `Cancel` a `true`, il framework procede con l'operazione e risponde con il risultato.

Se durante tutto il procedimento si verifica una qualsiasi eccezione, la chiamata termina con il seguente errore:

```
HTTP/1.1 404 Not Found
Content-type: text/plain; charset=utf-8

Unknown error: <testo dell'eccezione>
```

Al termine in ogni caso, tranne uno descritto in seguito, viene terminata la sessione.

6 Esempi di codice con personalizzazioni

In questo paragrafo vengono mostrati alcuni esempi di utilizzo dell'evento *OnWebApi* per intervenire sul comportamento predefinito del framework nella gestione della chiamata ad un metodo di Web API.

6.1 Autenticazione tramite header

Se si intende gestire l'autenticazione tramite header, si potrebbe globalizzare l'evento *OnWebApi* e implementarlo nel modo seguente:


```

event MyDocHelper.GlobalDocumentWebAPI(
    IDDocument Document // Source Document
    inout boolean Cancel // A boolean output parameter. If set to true it st...
)
{
    // Read username header
    IDMap requestHeaders = WebApiService.getHeaders()
    string username = requestHeaders.getValue("username")
    //
    // Find employee with this username
    int vIdEmployee = 0
    if (username != null)
    {
        select into variables (found variable)
        set vIdEmployee = EmployeeId
        from
        Employees // master table
        where
        LastName = username
    }
    //
    // Employee not authorized; abort the request
    if (vIdEmployee == 0)
    {
        Cancel = true
        WebApiService.setResponse(formatMessage("User |1 not found", username, ...
        ), 401, ...)
    }
}
}

```

Esempio di caricamento tramite header.

Nell'evento viene usata la funzione `WebApiService.GetHeaders` per recuperare gli header ricevuti nella richiesta. Inoltre si annulla la chiamata impostando il parametro `Cancel` a true.

Quando si annulla una chiamata, occorre indicare al framework ciò che deve essere scritto nella risposta e per farlo è necessario usare il metodo `SetResponse`.

In questo caso la risposta risulta la seguente:

```

HTTP/1.1 401 Unauthorized
User <username> not found

```

6.2 Filtro sulle proprietà

Una delle esigenze che potrebbe sorgere quando si attiva il servizio WebAPI su una classe già esistente è quella di nascondere alcune proprietà in modo che non vengano riportate nella risposta.

Supponiamo, ad esempio, di non voler mostrare il fornitore della classe Order. Per farlo è sufficiente implementare l'evento [OnSaveXMLProp](#) e annullare il parametro ExternalTag.

```
event Order.OnSaveXMLProp(  
    string InternalName // String parameter. The name of the property o...  
    inout string ExternalTag // String output parameter The name that the sy...  
    inout string Value // The value of the property that will be saved...  
    inout boolean UseElement // A boolean output parameter. Tells the system...  
)  
{  
    // Only for WebApi request  
    if (WebApiService.isWebApiRequest())  
    {  
        // hide shipper property  
        if (InternalName == "SHIPPER")  
            ExternalTag = ""  
    }  
}
```

Esempio di filtro sulle proprietà

Nella risposta non comparirà la proprietà con Tag SHIPPER.

6.3 Filtro sui figli

Un'altra possibile esigenza è quella di nascondere alcuni figli della gerarchia. Supponiamo ad esempio di avere la classe Order con due collection, OrderDetails e OtherCollection, e di non voler mostrare i figli contenuti in OtherCollection. Per farlo è sufficiente implementare l'evento [OnWebApi](#) e impostare la proprietà [Loaded](#) della collection a true.

```

event Order.OnWebAPI(
    inout boolean Cancel // A boolean output parameter. If set to true it st...
)
{
    // Only for request of search and load
    if (WebApiService.getMethod() == "GET")
    {
        // Set OtherCollection loaded
        OtherCollection.loaded = true
    }
}

```

Esempio di filtro sui figli

Il servizio WebAPI, dopo aver notificato l'evento, carica i figli e trovando la collection già caricata, seppure vuota, non la carica. Nella risposta non saranno quindi mostrati i figli di quella collection.

6.4 Ricerca personalizzata

Un'altra esigenza potrebbe essere quella di rispondere a delle ricerche in modo personalizzato.

Supponiamo ad esempio che ad una ricerca sugli ordini si desideri rispondere con una collection vuota. Per farlo è sufficiente popolare la proprietà *WebApiService.OutputCollection*.

```

event Order.OnWebAPI(
    inout boolean Cancel // A boolean output parameter. If set to true it st...
)
{
    IDMap parameters = WebApiService.getParameters()
    //
    // Only for request of search
    if (WebApiService.getMethod() == "GET" && parameters.length() > 0)
    {
        // Return an empty collection
        IDCollection empty of Order = new()
        empty.loaded = true
        WebApiService.outputCollection = empty
    }
}

```

Esempio di ricerca personalizzata

Il servizio WebAPI, dopo aver notificato l'evento, rileva che la collection è già stata caricata e pertanto non tenta di caricarla. Nella risposta sarà quindi riportata una collection vuota.

```
HTTP/1.1 200 OK
Content-type: text/xml; charset=utf-8

<IDCollection/>
```

6.5 Deep Linking

Il servizio WebAPI può essere usato anche per ottenere funzionalità di [Deep Linking](#). Supponiamo ad esempio di dover fare in modo che digitando l'URL <http://www.progamma.com/CRM/Order/43> si apra l'applicazione CRM nella quale sia mostrata direttamente la videata degli ordini sull'ordine con chiave 43. Per farlo è sufficiente implementare l'evento [OnWebApi](#) nel modo seguente:

```
event Order.OnWebAPI(
    inout boolean Cancel // A boolean output parameter. If set to true it st...
)
{
    if (WebApiService.getMethod() == "GET")
    {
        // Set userRole to bypass login page
        CRM.userRole = Administrator
        //
        // Show the desired form
        Orders.show([OpenAs])
    }
}
```

Esempio di deep linking

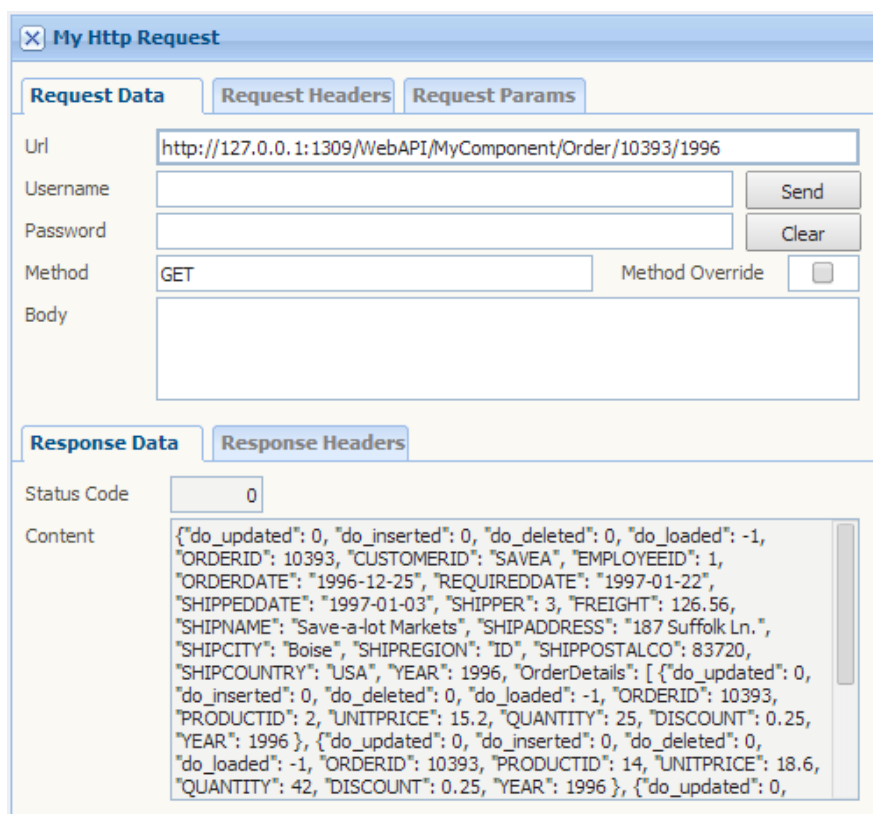
Se il servizio WebAPI, dopo aver notificato l'evento, rileva che non è stato chiamato il metodo [SetResponse](#), esegue una redirect sul documento predefinito in modo che l'applicazione risponda con l'interfaccia utente. È importante ricordarsi di gestire la proprietà UserRole perché altrimenti verrebbe visualizzata la videata di login.

7 Come testare una Web API

Poiché le Web API usano il protocollo HTTP per testare il funzionamento di una classe, con il servizio WebAPI è sufficiente usare un qualsiasi client in grado di effettuare richieste HTTP.

A questo scopo è disponibile un'applicazione Web realizzata con Instant Developer scaricabile da questo [indirizzo](#). L'applicazione contiene un'unica videata. Nella parte superiore vengono inseriti i dati della richiesta (URL, metodo, header, parametri, ecc.). Cliccando sul pulsante Send viene effettuata la richiesta.

I dati della risposta verranno mostrati nella parte inferiore della videata.



The screenshot shows a web application window titled "My Http Request". It has two main sections: "Request Data" and "Response Data".

Request Data Section:

- Request Data:** Selected tab. The "Url" field contains "http://127.0.0.1:1309/WebAPI/MyComponent/Order/10393/1996".
- Request Headers:** Unselected tab.
- Request Params:** Unselected tab.
- Username:** Empty text field.
- Password:** Empty text field.
- Method:** "GET" selected in a dropdown menu.
- Method Override:** A checkbox that is currently unchecked.
- Body:** Empty text area.
- Buttons:** "Send" and "Clear" buttons are located to the right of the Username and Password fields.

Response Data Section:

- Response Data:** Selected tab.
- Response Headers:** Unselected tab.
- Status Code:** A text field containing the value "0".
- Content:** A large text area displaying a JSON response. The content is:

```
{ "do_updated": 0, "do_inserted": 0, "do_deleted": 0, "do_loaded": -1, "ORDERID": 10393, "CUSTOMERID": "SAVEA", "EMPLOYEEID": 1, "ORDERDATE": "1996-12-25", "REQUIREDDATE": "1997-01-22", "SHIPDATE": "1997-01-03", "SHIPPER": 3, "FREIGHT": 126.56, "SHIPNAME": "Save-a-lot Markets", "SHIPADDRESS": "187 Suffolk Ln.", "SHIPCITY": "Boise", "SHIPREGION": "ID", "SHIPPOSTALCO": 83720, "SHIPCOUNTRY": "USA", "YEAR": 1996, "OrderDetails": [ { "do_updated": 0, "do_inserted": 0, "do_deleted": 0, "do_loaded": -1, "ORDERID": 10393, "PRODUCTID": 2, "UNITPRICE": 15.2, "QUANTITY": 25, "DISCOUNT": 0.25, "YEAR": 1996 }, { "do_updated": 0, "do_inserted": 0, "do_deleted": 0, "do_loaded": -1, "ORDERID": 10393, "PRODUCTID": 14, "UNITPRICE": 18.6, "QUANTITY": 42, "DISCOUNT": 0.25, "YEAR": 1996 }, { "do_updated": 0,
```

Videata per testare le WebAPI.

Qualora il valore ritornato non sia quello atteso si può compilare la parte server con il debug su file per verificare come è stata gestita la richiesta lato server.