



DelphiDay

italian conference

WiRL REST Library: uso avanzato

La più potente libreria REST per Delphi sfruttata al 100%



LUCA MINUTI



lucominuti.it



luca.minuti@gmail.com



dev.to/lminuti



github.com/lminuti



www.linkedin.com/in/lucominuti



DelphiDay

italian conference

11-12 Giugno 2024
Piacenza



wintech
italia

OPEN-SOURCE PROJECTS

github.com/Iminuti

WiRL

github.com/delphi-blocks/WiRL

Delphi SAML

github.com/EtheaDev/Delphi-SAML

OpenSSL

github.com/Iminuti/Delphi-OpenSSL



11-12 Giugno 2024
Piacenza





AGENDA

1. Context Injection
2. Filtri
3. MessageBody
4. Eccezioni
5. Validatori
6. Converter
7. Engine



WiRL






- 2.x: Neon, linux, Wizard, Swagger/OpenAPI, JSONP, RSA on JWT, UseUTCDate, Custom client engine
- 3.x: Proxy engine, generic configuration, exceptions mapping, SuperObject,
- 4.x: CORS, new WiRLClient, Converter, Garbage Collector, GraphQL, IWiRLTuple

<https://github.com/delphi-blocks/WiRL/releases>











RELEASE 4.6.0

High level features (since the 4.5.0 release)

-  Engines can decide if they want to handle a given URL (BasePath)
-  SinkPaths config to manage no-response paths (like favicon.ico, etc...)
-  Added "multipart/form-data" support to TWiRLClient
-  Experimental [SingleRecord] to get a single record (instead of an array) from a dataset
-  Ability to retrieve the response content as an object (Response.Content.AsType)

Other features, changes and bugfixes

-  Better management of client side exceptions (EWiRLClientProtocolException and EWiRLClientResourceException)
-  New TWiRLContent.AsType overload to "fill" an existing object
-  Resources can use absolute URLs ignoring those in TWiRLClientApplication and TWiRLClient
-  Introduced arrays support for WiRLConverters
-  Support for ISO date in WiRL.Data.Resolver
-  Updated JWT library to v3.3.1
-  Update Neon library to v3.0.0
-  Updated OpenAPI library to v2.2.0
- ☐ Fixed the attributes array parameter for message body reader and writer



Context Injection

1



INVERSION OF CONTROL

- Configurabilità: Dipendenze configurabili esternamente
- Dipendenza dall'Ambiente: Le dipendenze sono iniettate dall'esterno
- Modularità: Favorisce l'architettura modulare
- Testabilità: Migliora la facilità di test

Context injection

type

```
[Path('/myresource')]
```

```
TMyResource = class
```

```
private
```

```
  [Context] Application: TWiRLApplication;
```

```
public
```

```
  [GET]
```

```
  [Produces(TMediaType.APPLICATION_JSON)]
```

```
  function List([Context] Request: TWiRLRequest; [Context] Response: TWiRLResponse): string;
```

```
end;
```



OGGETTI PREDEFINITI

- **TWiRLServer**: the WiRL server
- **TWiRLEngine**: the WiRL engine
- **TWiRLApplication**: the current WiRL application
- **TWiRLRequest**: the current HTTP request
- **TWiRLResponse**: the current HTTP response
- **TWiRLURL**: the parsed URL of the HTTP request
- **TWiRLAuthContext**, **TWiRLSubject**: JWT objects



CONTEXT

- Nelle risorse
- Message body reader
- Message body writer
- Filters



CUSTOM INJECTION

- Si possono usare anche oggetti custom
 - Oggetti o data module che implementano la business logic
 - Un oggetto che fa il parsing di header HTTP
 - Un oggetto che trasporta informazioni dai filtri alle risorse
 - Comportamenti dipendenti da una configurazione
 - ...

<https://github.com/delphi-blocks/WiRL/wiki/Context-Injection>



CICLO DI VITA

- Gli oggetti sono creati da una **factory**
- Di norma sono distrutti da WiRL al termine della stream HTTP
- È possibile disabilitare questo comportamento con l'attributo **Singleton**

demo time





FILTRI

2



FILTRI

- I filtri possono intercettare richiesta e risposta e:
 - Modificare gli headers
 - Modificare il corpo
 - Fare logging
 - Controllare permessi e privilegi
 - Sollevare una eccezione e bloccare l'esecuzione della API
 - Saltare completamente la risorsa rispondendo al suo posto



Tipologie

- Filtri sulla richiesta
 - Partono prima che il codice associato alla risorsa sia stato eseguito
 - Permettono di modificare la richiesta
- Filtri sulla risposta
 - Partono dopo la risorsa
 - Permettono di modificare la risposta

FILTRO SULLA RISPOSTA

type

```
TResponsePoweredByFilter = class(TInterfacedObject, IWiRLContainerResponseFilter)
public
    procedure Filter(AResponseContext: TWiRLContainerResponseContext);
end;
```

implementation

```
procedure TResponsePoweredByFilter.Filter(AResponseContext: TWiRLContainerResponseContext);
begin
    if AResponseContext.Response.StatusCode >= 500 then
        AResponseContext.Response.HeaderFields['X-Powered-By'] := 'DataSnap' // ;-)
    else
        AResponseContext.Response.HeaderFields['X-Powered-By'] := 'WiRL';
end;
```

initialization

```
WiRLFilterRegistry.Instance.RegisterFilter<TResponsePoweredByFilter>;
```




PRE-MATCHING

- Normalmente i filtri partono solo se la risorsa è stata individuata (Path, HttpMethod, Accept-*, ...)
- L'attributo **[PreMatching]** permette di eseguire il filtro prima della ricerca della risorsa



NAME BINDING

- I filtri vengono chiamati per tutte le risorse. Possono essere selezionati tramite:
 - Configurazione (App)
 - Controlli sul filtro stesso
 - Name-Binding (è possibile creare un attributo custom che associa il filtro a una o più risorse)

NAME BINDING

type

```
// I declare the new attribute
[NameBinding]
ContentEncodingAttribute = class(TCustomAttribute);

// I use the new attribute to decorate the filter
// than I must decorate the resource method with the same attribute
[ContentEncoding]
TResponseEncodingFilter = class(TInterfacedObject, IWiRLContainerResponseFilter)
private
    const ENC_GZIP = 'gzip';
    const ENC_DEFLATE = 'deflate';
    const ENC_IDENTITY = 'identity';
public
    procedure Filter(AResponseContext: TWiRLContainerResponseContext);
end;
```



PRIORITÀ

- Nel caso esistano più filtri vengono eseguiti in base alla priorità selezionata:
 - AUTHENTICATION: 1000
 - AUTHORIZATION: 2000
 - HEADER_DECORATOR: 3000
 - ENTITY_CODER: 4000
 - USER: 5000 (default)

demo time





MESSAGE
BODY R/W

3



READER & WRITER

- Una delle caratteristiche principali di WiRL è quella di scrivere i metodi delle risorse dimenticandosi che si sta scrivendo una API ReST
- Una delle chiavi di questo meccanismo sono i Message Body Reader e Writer



READER & WRITER

- Message Body Reader:
 - Trasformano il contenuto della richiesta nell'oggetto: JSON in TPerson, Lista di stringhe in TString, dati binary in un TStream, ...
- Message Body Writer:
 - Trasformano l'output della risposta in un flusso di dati: TDataSet in JSON, TPerson on XML, ...

MESSAGE BODY

```
THelloWorldResource = class
public
  function PostOrder(AOrderProposal: TOrderProposal): TOrder;
end;
```

```
[Path('/helloworld')]
THelloWorldResource = class
public
  [POST]
  [Path('/order')]
  [Consumes(TMediaType.APPLICATION_JSON)]
  [Produces(TMediaType.APPLICATION_JSON)]
  function PostOrder([BodyParam] AOrderProposal: TOrderProposal): TOrder;
end;
```



READER & WRITER

- In WiRL ci sono decine di Reader e Writer:
 - TWiRLSimpleTypesProvider: tipi semplici, interi, double, ...
 - TWiRLValueTypesProvider: array e record
 - TWiRLObjectProvider: oggetti, DataSet, StringList, ... (attraverso neon)
 - TWiRLJSONValueProvider: TJSONObject
 - TWiRLStreamProvider: discendenti di TStream
 - TWiRLMultipartFormDataProvider: TMultipartFormData



READER & WRITER

- e ancora:
 - TArrayDataSetProvider: array di DataSet
 - TDataSetWriterXML: DataSet to XML
 - TDataSetWriterCSV: DataSet to CSV
 - TWiRLFireDACAdaptedDataSetProvider: DataSet di FireDAC
 - TWiRLFireDACDataSetArrayProvider: array di DataSet di FireDAC
 - ...



NEON

- Per gli oggetti viene utilizzato NEON
- Riesce a serializzare e deserializzare oggetti complessi
- Estremamente flessibile:
 - Nome delle proprietà (NeonProperty)
 - Quando visualizzarle (NeonInclude)
 - Gestione dei dati binary (NeonFormat)
 - Gestione di array disomogenei (NeonItemFactory)

<https://github.com/paolo-rossi/delphi-neon>

MESSAGE BODY - NEON

```
FApp := TWiRLServer.Create(nil);  
FApp  
  // ...  
  .Plugin.Configure<TWiRLConfigurationNeon>  
    .SetUseUTCDate(True)  
    .SetVisibility([mvPublic, mvPublished])  
    .SetMemberCase(TNeonCase.ScreamingSnakeCase)  
    .BackToApp;
```



CUSTOM

- È possibile creare Reader e Writer personalizzati
- Oppure serializzatori / deserializzatori per neon (solo se il formato è JSON)

MESSAGE BODY

type

```
[Consumes(TMimeType.APPLICATION_JSON)]  
[Produces(TMimeType.APPLICATION_JSON)]  
TWiRLMyProvider = class(TMessageBodyProvider)  
public  
    function ReadFrom(...): TValue; override;  
    procedure WriteTo(...); override;  
end;
```

implementation

```
// ...
```

initialization

```
TMessageBodyReaderRegistry.Instance.RegisterReader(...);  
  
TMessageBodyWriterRegistry.Instance.RegisterWriter(...);
```

demo time





ECCEZIONI

4



ECCEZIONI

- WiRL cattura automaticamente le eccezioni generate nelle risorse e le restituisce in JSON
- Esiste una eccezione generica `EWiRLWebApplicationException` che permette di personalizzare messaggio e status code e alcune più specifiche:
 - `EWiRLBadRequestException` (400), `EWiRLNotAuthorizedException` (401), `EWiRLNotFoundException` (404), `EWiRLServerException` (500)



STATUS CODE

- 2xx success
 - 200 OK, 201 Created, 202 Accepted, 203 Non-Authoritative Information, 204 No Content, 206 Partial Content
- 3xx redirection
 - 301 Moved Permanently, 304 Not Modified
- 4xx client errors
 - 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 409 Conflict
- 5xx server errors
 - 500 Internal Server Error, 501 Not Implemented, 503 Service Unavailable

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

MESSAGE BODY

```
raise Exception.Create('User Error Message');
```

```
{  
  "status": 500,  
  "exception": "Exception",  
  "message": "User Error Message"  
}
```

```
raise EWiRLServerException.Create('User Error Message', 501,  
  TValuesUtil.MakeValueArray(  
    Pair.S('name', 'value'),  
    Pair.N('code', 42)  
  ));
```

```
{  
  "status": 501,  
  "exception": "EWiRLServerException",  
  "message": "User Error Message",  
  "data": {  
    "name": "value",  
    "code": 42  
  }  
}
```



CUSTOM

- È possibile personalizzare la risposta in caso di errore
- Ad ogni eccezione è possibile associare un messaggio
- Sia per le proprie eccezioni che per quelle di librerie di terze parti
- La risposta può essere in qualsiasi formato (non necessariamente JSON)

EXCEPTION

type

```
TTestExceptionMapper = class(TWiRLExceptionMapper)
public
    procedure HandleException(AExceptionContext: TWiRLExceptionContext); override;
end;
```

implementation

```
// ...
```

initialization

```
TwIRLExceptionMapperRegistry.Instance.RegisterExceptionMapper<TTestExceptionMapper, ETestException>();
```

demo time





CONVERTER

5



CONVERTER

- I converter sono usati per leggere (WiRLServer) e scrivere (WiRLClient) i parametri che non sono nel body del messaggio: PathParam, QueryParam, HeaderParam, ecc.
- Separatore decimale, formato delle date, enumerativi, ecc.

CONVERTER

```
FServer.AddEngine<TWiRLEngine>('/rest')  
    .SetEngineName('RESEngine')  
    .AddApplication('/app')  
    .SetResources('*')  
    .SetFilters('*')  
    .Plugin.Configure<IWiRLFormatSetting>  
        .AddFormat(TypeInfo(TDate), TWiRLFormatSetting.MDY)  
    .ApplyConfig
```




CONVERTER

- Esistono convertitori per:

- **Date:** TISODateConverter, TUnixDateTimeConverter, TUnixDateConverter, TYMDDateConverter
- **DateTime:** TISODateTimeConverter
- **Float:** TDefaultFloatConverter,
- **Integer:** TDefaultIntegerConverter
- **Int64:** TDefaultInt64Converter
- **Currency:** TDefaultCurrencyConverter
- **Boolean:** TDefaultBooleanConverter
- **String:** TDefaultStringConverter
- **Enum:** TDefaultEnumConverter



CONVERTER

- È possibile creare convertitori personalizzati per:
 - Gestire enumerativi
 - Array
 - Record
 - Altri tipi per cui non esiste un convertitore predefinito

demo time





VALIDATORI

6



VALIDATORI

- I validatori definiscono dei vincoli (NotNull, Max, Min, ...)
- Possono essere associati ad un parametro (PathParam, QueryParam, BodyParam, ...)
- Se il vincolo non è rispettato la chiamata fallisce



VALIDATORI

- Validatori predefiniti:
 - Min: valore minimo (applicato ad un numero)
 - Max: valore massimo (applicato ad un numero)
 - NotNull: parametro obbligatorio
 - Pattern: deve superare una espressione regolare
 - Size: numero di caratteri

VALIDATORI

```
[Path('validator')]
```

```
TValidatorDemoResource = class
```

```
public
```

```
[GET, Path('/double/{AValue}'), Produces(TMediaType.TEXT_PLAIN)]
```

```
function Double([PathParam][Max(50), Min(1, 'Too small')] AValue: Integer): Integer;
```

```
[GET, Path('/concat?s1={s1}&s2={s2}'), Produces(TMediaType.TEXT_PLAIN)]
```

```
function Concat(
```

```
    [QueryParam('email'), Pattern('.+@.+\\.+.+', 'E-Mail is not valid')] EMail: string;
```

```
    [QueryParam('name'), NotNull('Name required')] Name: string): string;
```

```
// Crazy test with a lot of attributes
```

```
[GET, Path('/test?i={i}'), Produces(TMediaType.TEXT_PLAIN)]
```

```
function Test([Max(10), DefaultValue('0'), Pattern('1'), Size(2, 2), QueryParam('i')] i: Integer): Integer;
```

```
end;
```



CUSTOM

- È possibile creare validatori personalizzati:
 - Si definisce un nuovo attributo
 - Una classe che implementa l'interfaccia `IConstraintValidator<T>`
 - Infine si implementa il metodo `IsValid` con i controlli necessari
 - Si registra la classe

VALIDATORI

```
HasNameAttribute = class(TCustomConstraintAttribute)
end;
```

```
THasNameValidator = class(TInterfacedObject, IConstraintValidator<HasNameAttribute>)
public
    procedure Initialize(AHasNameAttribute: HasNameAttribute);
    function IsValid(AValue: TValue; Context: TWiRLContext): Boolean;
end;
```

implementation

```
procedure THasNameValidator.Initialize(AHasNameAttribute: HasNameAttribute);
begin ... end;
```

```
function THasNameValidator.IsValid(AValue: TValue; Context: TWiRLContext): Boolean;
begin ... end;
```

initialization

```
TWiRLValidatorRegistry.Instance.RegisterValidator<THasNameValidator>;
```

demo time





ENGINE

7



ENGINE

- La parte server di WiRL è composta da un server HTTP (TWiRLServer) a cui normalmente viene agganciato un motore ReST (TWiRLEngine)
- Esistono però vari altri engine:
 - TWiRLhttpEngine: permette di gestire richieste tramite eventi
 - TWiRLFileSystemEngine: fornisci file statici
 - TWiRLProxyEngine: proxy HTTP



FILE SYSTEM ENGINE

- Fornisce file statici
- Permette di evitare l'uso di CORS
- Non gestire cache, sicurezza o altro
- NON USARE IN PRODUZIONE

FILE SYSTEM ENGINE

```
FServer := TWiRLServer.Create(nil);
```

FServer

```
// ReST engine configuration
.AddEngine<TWiRLEngine>('/rest')
.SetEngineName('WiRL Template')

// Application configuration
.AddApplication('/default')
.SetAppName('Default')
.SetResources('*');
```

FServer

```
// File sistem engine configuration
.AddEngine<TWiRLFileSystemEngine>('/')
.SetRootFolder('{AppPath}\wwwroot');
```



PROXY ENGINE

- Permette di vedere delle risorse remote come se provenissero dallo stesso dominio del server
- NON USARE IN PRODUZIONE

PROXY ENGINE

```
FServer := TWiRLServer.Create(Self);
```

```
FServer.AddEngine<TWiRLProxyEngine>('/delphi')
```

```
.SetRemoteUrl('https://www.delphiday.it/');
```

```
FServer.AddEngine<TWiRLProxyEngine>('/assets')
```

```
.SetRemoteUrl('https://www.delphiday.it/assets');
```

```
FServer.AddEngine<TWiRLEngine>('/rest')
```

```
.SetEngineName('RESEngine')
```


demo time





THANK YOU