

An aerial photograph of a historic Italian city, likely Florence, showing a dense cluster of buildings with terracotta roofs, a winding river, and a large square with a prominent building. The image is used as a background for the text.

# DELPHIDAY

italian conference

WiRL

Server-Sent event, Engine e molto altro



**LUCA MINUTI**



[luccaminuti.it](https://luccaminuti.it)



[luca.minuti@gmail.com](mailto:luca.minuti@gmail.com)



[dev.to/lminuti](https://dev.to/lminuti)



[github.com/lminuti](https://github.com/lminuti)



[www.linkedin.com/in/luccaminuti](https://www.linkedin.com/in/luccaminuti)

# DELPHIDAY

italian conference

9-10 Giugno 2026  
Piacenza



**wintech**  
italia



**OPEN-SOURCE** PROJECTS

[github.com/lminuti](https://github.com/lminuti)

**WiRL**

[github.com/delphi-blocks/WiRL](https://github.com/delphi-blocks/WiRL)

**Delphi Blocks**

[github.com/delphi-blocks/blocks](https://github.com/delphi-blocks/blocks)

**MCPCConnect**

[github.com/delphi-blocks/MCPCConnect](https://github.com/delphi-blocks/MCPCConnect)

# DELPHIDAY

italian conference

9-10 Giugno 2026  
Piacenza



**wintech**  
italia



# AGENDA

---

1. **Server Sent Events (SSE): cos'è e come implementarlo**
2. **SSE per notifiche a client e chunk encoding**
3. **Engine per implementare altri protocolli JRPC, MCP, GraphQL**
4. **Integrare sistemi di logging con Logify**
5. **Le classi “nascoste” di WiRL: WiRL.MB.SingleRecord, WiRL.Engine.Proxy, WiRL.Data.Resolver**



SSE

1



# SSE – Server-sent events

- Permette di **inviare degli eventi dal server al client**
- Il client si collega ad un particolare endpoint del server mantenendo la connessione **perennemente aperta**
- Anche in caso di disconnessione (per problemi di rete o quant'altro) il client si deve occupare di ripristinarla non appena possibile



# SSE – Server-sent events

- Il server deve rispondere con mime type: `text/event-stream`
- Ogni evento può avere i seguenti campi:
  - **Data**: il corpo del messaggio
  - **Id**: identificativo del messaggio
  - **Event**: nome dell'evento
  - **Comment**: commento/note
  - **Retry**: tempo dopo il quale riconnettersi (in caso di problemi)
  - **Altro**: viene ignorato

# Esempi

HTTP/1.1 200 OK

Content-Type: text/event-stream

data: This is the first message.

data: This is the second message, it  
data: has two lines.

id: 42

data: This is the third message.

—

event: add

data: 73857293

event: remove

data: 2153



# Specifiche SSE

```
stream      = [ bom ] *event
event       = *( comment / field ) end-of-line
comment     = colon *any-char end-of-line
field       = 1*name-char [ colon [ space ] *any-char ] end-of-line
end-of-line = ( cr lf / cr / lf )

; characters
lf          = %x000A ; U+000A LINE FEED (LF)
cr          = %x000D ; U+000D CARRIAGE RETURN (CR)
space       = %x0020 ; U+0020 SPACE
colon       = %x003A ; U+003A COLON (:)
bom         = %xFEFF ; U+FEFF BYTE ORDER MARK
name-char   = %x0000-0009 / %x000B-000C / %x000E-0039 / %x003B-10FFFF
              ; a scalar value other than U+000A LINE FEED (LF), U+000D
              CARRIAGE RETURN (CR), or U+003A COLON (:)
any-char    = %x0000-0009 / %x000B-000C / %x000E-10FFFF
              ; a scalar value other than U+000A LINE FEED (LF) or U+000D
              CARRIAGE RETURN (CR)
```

# Implementazione client JavaScript

```
var source = new EventSource('/rest/app/events');  
source.onmessage = function (event) {  
    alert(event.data);  
};
```



# WiRL

- Creazione di un metodo come di consueto
- `TMediaType.TEXT_EVENT_STREAM`
- Il metodo deve restituire `TWiRLSSEResponse`
- Che al suo interno avrà un metodo anonimo che si occuperà di inviare gli eventi
- Il metodo anonimo deve essere `ThreadSafe`



- All'interno del metodo anonimo si possono inviare eventi tramite questi metodi:
  - procedure Write(const AValue: string);
  - procedure Write(const AEvent, AValue: string);
  - procedure Write(const AEvent, AValue: string; ARetry: Integer);
  - procedure WriteComment(const AValue: string);

# Implementazione con WiRL

```
function TMyResource.ServerSideEvents(const ATag: string): TWiRLSSEResponse;  
begin  
    Result := TWiRLSSEResponse.Create(  
        procedure (AWriter: IWRLSSEResponseWriter)  
            var  
                LMessage: string;  
            begin  
                // Continua finché la connessione è attiva  
                while AWriter.Connected do  
                    begin  
                        // Legge un messaggio dalla coda  
                        LMessage := MessageQueue.PopItem;  
                        if LMessage <> '' then  
                            // Se lo trova lo invia al client  
                            AWriter.Write(LMessage);  
                    end;  
                end  
            );  
        end;  
end;
```



# demo time





# Chunks

- Transfer encoding di tipo chunked
- Permette di **inviare i dati a blocchi**
  - dimensione totale del contenuto non è nota in anticipo
  - streaming di dati in tempo reale
  - migliorare i tempi di risposta percepiti
  - trasferimenti di file di grandi dimensioni



# Chunks

- L'header Transfer-Encoding deve essere **chunked**
- L'header Content-Length deve essere omesso
- I dati devono essere inviati a pacchetti verso il client
- Il client può renderizzarli o processarlo in qualche modo man mano che arrivano

# Esempio

HTTP/1.1 200 OK

Content-Type: text/plain

Transfer-Encoding: chunked

7\r\n

Welcome\r\n

1c\r\n

to Mozilla Developer Network\r\n

0\r\n

\r\n



# Chunks – WiRL

- Implementazione molto simile a SSE
- Metodo restituisce `TWiRLChunkedResponse`
- C'è una procedura anonima che invia i chunk man mano che arrivano
- I chunks possono essere anche binari



# Implementazione con WiRL

```
function TMyResource.Chunks(ANumOfChunks: Integer): TWiRLChunkedResponse;  
begin  
    Result := TWiRLChunkedResponse.Create(  
        procedure (AWriter: IWiRLResponseWriter)  
            var  
                LCounter: Integer;  
            begin  
                // Invia i dati in ANumOfChunks chunks  
                for LCounter := 1 to ANumOfChunks do  
                    begin  
                        // Invia il singolo chunk  
                        AWriter.Write(IntToStr(LCounter));  
                        // Simula l'attesa necessaria per ottenere il dato successivo  
                        Sleep(1000);  
                    end;  
                end  
            );  
        end;  
    end;  
end;
```

# demo time





# Engine

# 2



# Engine

- WiRL è composto 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 system engine configuration
```

```
    .AddEngine<TWiRLFileSystemEngine>('/')
```

```
    .SetRootFolder('{AppPath}\wwwroot');
```

# demo time





# Proxy Engine

- Permette di vedere delle risorse remote come se provenissero dallo stesso dominio del server
- NON USARE IN PRODUZIONE (a meno di traffico limitato)

# 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('RESTEngine')
```

# demo time







# MCP/JSON-RPC Engine

- Si può creare un server JSON-RPC con **MCPConnect** e collegarlo a WiRL
- La configurazione avviene tramite il TJRPCServer
- Poi si crea il TMCPEngine e lo si collega al TJRPCServer

# MCP Engine

```
FJRPCServer := TJRPCServer.Create(Self);  
// Si configura il server come al solito  
  
// Create http server  
FWiRLServer := TWiRLServer.Create(nil);  
  
// Server configuration  
FWiRLServer  
    .SetPort(StrToIntDef(PortNumberEdit.Text, 8080))  
    // ReST Engine configuration  
    .AddEngine<TWiRLRESEngine>('/rest')  
    .SetEngineName('WiRL Demo')  
  
    // MCP Engine configuration  
    .AddEngine<TMCPEngine>('/mcp')  
    .SetServer(FJRPCServer);
```

**demo time**





# Custom Engine

- È possibile creare degli Engine personalizzati
- Per esempio SOAP, GraphQL, ...
- Sono sufficienti i seguenti passaggi:
  - derivare da `TWiRLCustomEngine`
  - ed implementare il metodo `HandleRequest`

# demo time





# Classi “nascoste”

# 3



# SingleRecord

- Nel caso un metodo restituisca un DataSet WiRL è in grado di serializzarlo automaticamente (tramite **Neon**)
- Normalmente crea **un array di oggetti**, dove ogni oggetto corrisponde ad un record
- Questo anche se il DataSet ha un solo record
- Nel caso in cui si sia un solo record è possibile forzare WiRL ha serializzare il DataSet con un oggetto invece che un array tramite l'attributo **[SingleRecord]**



# Data Resolver

- Quando si lavora WiRL le operazioni CRUD si risolvono nel
- WiRL ha una classe TWiRLResolver che permette di semplificare la persistenza dei DataSet




# demo time



An aerial photograph of a city, likely Florence, Italy, showing a dense urban landscape with a river (the Arno) winding through it. A large, historic square (Piazza della Signoria) is visible in the lower center, featuring a prominent building with arches. The image is overlaid with a semi-transparent dark brown filter.

# DELPHIDAY

  
italian conference

## THANK YOU