TUTORIAL · JUNE 28, 2023

TMS WEB Core with Tabulator

Display and edit tabular data with Tabulator and TMS WEB Core

By Dr. Holger Flick

Displaying and editing tabular data is one of the strengths developers think of when they hear the programming language Delphi. Database-driven desktop applications need proper ways to display, filter, sort, and edit records. As TMS WEB Core is designed for Delphi developers to create Web applications, integration of grids is an essential part.

> **Source code**
> You find the source code for this example on GitHub: <https://github.com/holgerflick/hc-tabulatorjs-examples>

TMS provides three key components:

1. `TWebDBGrid` is very much a Web implementation of `TDBGrid`. However, it is very limited with regards to responsive web design.
2. `TWebResponsiveGrid` to provide a responsive solution for `TDBGrid`. Truly, the best component to look into when you do not want to take the step using a CSS framework like Bootstrap but need to present your Web application on multiple devices with different display sizes.
3. `TWebTableControl` is designed for usage in Bootstrap Web applications. Thus, it's number one strong point is responsive web design. However, there is lack of proper editing support and filtering functionality.

This tutorial looks at an alternative from the JavaScript world which can easily be integrated into your TMS WEB Core application.

> **Hint**
> We will **not** build a component and the amount of JavaScript will kept to an **absolute minimum**. TMS WEB
>
> Core is designed for Delphi developers and thus the amount of embedded JavaScript will be kept to an absolute minimum. ``` ```delphi asm // JavaScript code end; ``` ``` All JavaScript code will rather be part of Web design and will not clutter your Delphi source code. This way, you can work in cooperation with a Web designer to create the desired layout.

Part 1: Building a simple example

<iframe width="560" height="315" src="https://www.youtube-nocookie.com/embed/qOp_KDRXuWs" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>

In this example we will use the Tabulator framework. You can download it here.

1. Load the JavaScript library in the project HTML file of your application.

```
   <link href="https://unpkg.com/tabulator-tables/dist/css/tabulator.min.css"
      rel="stylesheet">
   <script type="text/javascript"
      src="https://unpkg.com/tabulator-tables/dist/js/tabulator.min.js">
   </script>
```

1. The application will use Web layout specified in HTML. Thus, add the following HTML to the main form:

```html title="uFrmMain.html" linenums="1" <html> <head> <meta http-equiv="Content-type" content="text/html; charset=utf-8" /> <title>TMS Web Project</title> <style> </style> </head> <body> <div class="container p-5"> <p class="display-1">Tabulator Basic Example</p> <div class="row"> <div class="col-10"> <div id="example-table"></div> </div> <div class="col-2"> <button class="btn btn-primary" id="btnShow">Show data</button> </div> </div> </div>

<script> function showTable(tabData) { var table = new Tabulator("#example-table", { data: tabData, //assign data to table columns: [ { title: "Name", field: "Name", editor: "input", sorter: "string" } , { title: "Age", field: "Age", editor: "number", sorter: "number" } , { title: "Job", field: "Job", editor: "input", sorter: "string" } , { title: "Birthday", field: "Birthday", editor: "date", sorter: "date" } ] }); } </script> </body> </html>
```

```
### Linking the control to HTML

As Tabulator is a JavaScript component, it needs to be linked to your HTML layout. This is done using the `id` of the HTML control it is supposed to replace. Instead of the HTML control, the table will be rendered.
```

 Tabulator Basic Example   <button class="btn btn-primary" id="btnShow">Show data</button>

```
In this example, the `<DIV>` tag with the `id` of `example-table` will be replaced with the data grid.

### Creating a new data grid
```

<script> function showTable(tabData) { var table = new Tabulator("#example-table", { data: tabData, //assign data to table columns: [{ title: "Name", field: "Name", editor: "input", sorter: "string" } , { title: "Age", field: "Age", editor: "number", sorter: "number" } , { title: "Job", field: "Job", editor: "input", sorter: "string" } , { title: "Birthday", field: "Birthday", editor: "date", sorter: "date" }] }); } </script>

```
In order to create the data grid, we need JavaScript. However, it is not part of your Delphi code. Instead, it is directly copied from the Tabulator documentation and becomes part of the HTML. This way, your Delphi code stays uncluttered and easy to maintain without JavaScript knowledge.

### Column definition
```

columns: [{ title: "Name", field: "Name", editor: "input", sorter: "string" } , { title: "Age", field: "Age", editor: "number", sorter: "number" } , { title: "Job", field: "Job", editor: "input", sorter: "string" } , { title: "Birthday", field: "Birthday", editor: "date", sorter: "date" }]

```
Each column is specified in the `columns` property of the definition of the control. JSON is
used in JavaScript to express this. Look [here](https://tabulator.info/docs/5.5/columns) for all
customization options for columns.

### Defining data

Each of the columns that you want to display in a grid needs to match a property in a JSON object.
In order to supply multiple records, an array of JSON objects is being used.

TMS WEB Core makes this straightforward as you can use a record which can be automatically converted
into a JSON object.
```

type TPerson = record Name: String; Age: Integer; Job: String; Birthday: String; end;

```
You can just pass an `Array of TPerson` to a JavaScript function and TMS WEB Core will convert it
to the necessary JSON representation.
```

// call JavaScript function that you copied from the documentation asm showTable(LData); end;

```
### Creating example data
```

procedure TForm1.btnShowClick(Sender: TObject); var LData: array of TPerson;

begin // initialize the array SetLength(LData, 2);

// assign first person LData[0].Name := 'Holger Flick'; LData[0].Age := 27; LData[0].Job := 'Evangelist'; LData[0].Birthday := TBclUtils.DateToISO(EncodeDate(1996, 2, 10));

// assign second person LData[1].Name := 'Bruno Fierens'; LData[1].Age := 28; LData[1].Job := 'Innovator'; LData[1].Birthday := TBclUtils.DateToISO(EncodeDate(1995, 2, 14));

// call function end;

```
## Part 2: Retrieving data from a Web service

This example will feature additional column types. Also, you will learn how to offer filtering and
sorting easily.

<iframe width="560" height="315" src="https://www.youtube-nocookie.com/embed/NIVRDtCJrOM" ti-
tle="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; en-
crypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>

The application built will look like this and shows the features this demo will focus on with
regards to the Tabulator control. Obviously, requesting the data from an external, public API is
also an important part that we need to cover.

![Screenshot](/tutorials/images/tab-03.png)

1. Images can be presented easily.
1. Text boxes to filter. The response is immediate and executed on the client.
1. Sort indicators to switch the sort order of a column and designate it as the sort criterion.

### Apple iTunes Search API

Instead of using data from a backend that is created by us, we use an API that is available for
free and for everybody. Accessing it is very easy as no authentication is required. You also do
not need to sign up with Apple in order to use it. Each request is issues using an HTTP GET request
and all parameters are encoded in its URL.

The documentation for the API is available [:link: here](https://developer.apple.com/li-
brary/archive/documentation/AudioVideo/Conceptual/iTuneSearchAPI/index.html). The quality of the
documentation is not bad, but it is also not all-encompassing. You will have to do a lot of
try-and-error to investigate what the API returns when you issue a certain search request.

!!! tip "TMS RestInsight"
    TMS Software offers a great tool for free which is available for Windows and Apple macOS
that helps you to 'browse' public APIs and learn how to query the results that you desire. Go to
<https://www.tmssoftware.com/site/blog.asp?post=1087> to learn more.

### Nothing XData specific

This example uses components that are part of TMS WEB Core. None of the components used require a
TMS XData license. TMS XData is a tool to implement a backend. You use it to create your own web
service API. In this example, we are going to use an API that already exists. This approach is
truly generic and can be used for any other API that you might encounter.

!!! note
    This example will **not** use XData-specific controls. Thus, you can apply this example to
**any Web service** that uses JSON payloads.

### Requesting a movie title

To get started, we are going to hard-code the information that is being requested. You can make it
your own exercise for the day to add a text box and a button that will allow end users to inquire
different movies.

The URL to be used has the following structure. You learn this by reading the documentation. This
is nothing you **can know or learn** as a Delphi developer. Each API is different.
```

https://itunes.apple.com/search?term=<Search Term>&country=us&entity=movie

```
1. The base URL which refers to the Apple iTunes Search API.
1. The first query parameter specifies the search terms.
1. We limit the search to the US <ú<ø
1. Only movies are returned.

!!! warning "Search Terms"
    Search terms that are passed to an API this way need to be encoded to be applicable for usage
inside of a URL.
    That means you cannot have spaces in your search terms or use special characters like `/` or
`'`, for example.
    This is why we start with a fixed search term to reduce the complexity of this example.

### Component for Web requests

We use the `TWebHttpRequest` component to retrieve data from the internet. This Apple API is no
different as it provides an HTTPS endpoint and can thus be treated like any other web site.

After dropping the component onto the form, you can set its properties as follows:

![Screenshot](/tutorials/images/tab-01.png)

Also, we need a button on the form which we link to the HTML provided.

![Screenshot](/tutorials/images/tab-02.png)

### HTML design

Again, we use HTML web design instead of creating our design using the Delphi form designer. Only
a button is created.
As shown in part 1, we need to provide space for the Tabulator component. The tag named
`example-table` will do just that.
```

<body> Tabulator Web Service Example   <button class="btn btn-primary" id="btnShow">Show data</button>   </body>

```
Set the `ElementID` property of the `TWebButton` control to `btnShow` and the design part of the
application is done.

### Extending the form class

The form will be extended with one method to request the data from the Web service. As we will use
the new `await` feature from TMS WEB CORE, the custom attribute `[async]` needs to be used.
```

type TFrmMain = class(TWebForm) btnShow: TWebButton; Request: TWebHttpRequest; procedure btnShowClick(Sender: TObject); private { Private declarations } FData: TJSArray; public { Public declarations } procedure ProcessResult(AResponse: TJSXMLHttpRequest);

[async] procedure RequestData; end;

```
### Requesting the data

We use `await` to wait for the server to respond. The response will be stored in `LResponse`. Even
when getting a response
from the server does not mean that the request was successful. We need to inspect the HTTP response
code which can be read using the `Status` property. If it is equal to `200`, the request was
successful. If the response was received, we call `ProcessResult` passing it as a parameter.
```

procedure TFrmMain.RequestData; var LResponse: TJSXMLHttpRequest;

begin // wait for data request to be received LResponse := await(TJSXMLHttpRequest, Request.Perform);

// proceed if status code is 200 if LResponse.Status = 200 then begin // process result and show grid ProcessResult(LResponse); end; end;

```
### Processing the data

Passing
```

procedure TFrmMain.ProcessResult(AResponse: TJSXMLHttpRequest); begin // Response can be used directly! console.log(AResponse.response);

// assign data to global variable self.FData := TJSArray(TJsObject(AResponse.response)['results']);

// pass data to grid and display {$IFDEF pas2js} asm showTable(this.FData) end; {$ENDIF} end;

```
### Presenting the data
```

<script> function showTable(data) { var table = new Tabulator("#example-table", { data:data, columns: [{ title: "Cover", field: "artworkUrl100", formatter: "image" } , { title: "Title", field: "trackName", headerFilter: "input", editor: true } , { title: "Release", field: "releaseDate", formatter: "datetime", formatterParams: { inputFormat: "iso", outputFormat: "dd/MM/yyyy" } } , { title: "Genre", field: "primaryGenreName", headerFilter: "list", headerFilterParams: { autocomplete: true, valuesLookup: true, clearable: true } }] }); } </script>

```
![Cover](/books/images/cover_3_250.png)
!!! tip "TMS WEB Core Book, 2nd Edition"
    If you want to learn more about TMS WEB Core, please have a look at my book that is designed for beginners that want to learn about Web development with Delphi as well as advanced developers that want to get the most out of TMS WEB Core. Examples in the book go in much more detail and as all basics are covered as well, you can always go back to read about it again. Something these tutorials simply cannot provide.

    Go to <https://flixengineering.com/books> to learn more. If you are located in the US, simply order at [Amazon](https://www.amazon.com/dp/B0C2RX8NXK).
```

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —