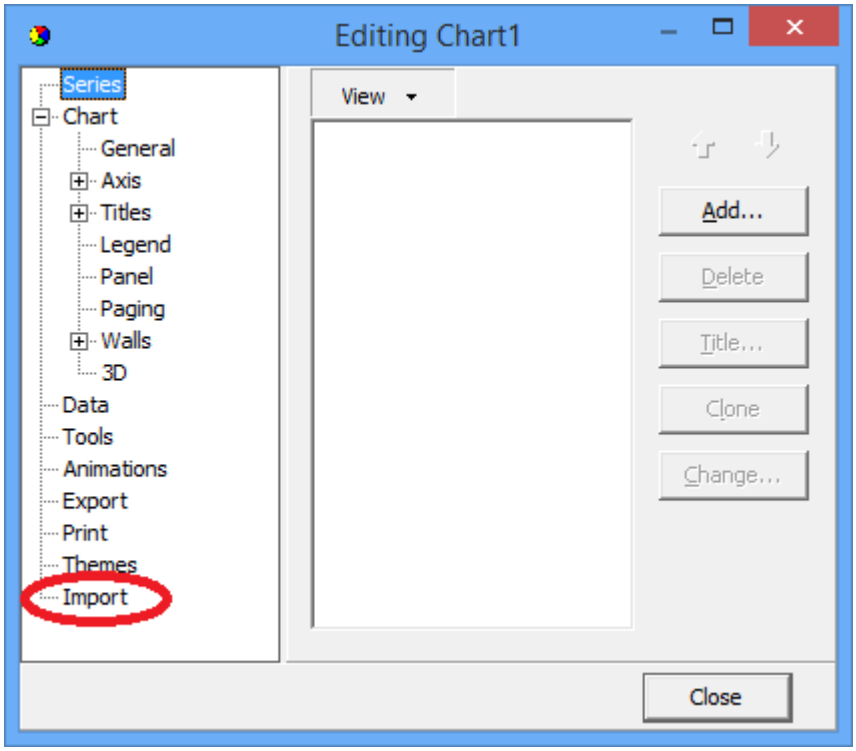
TeeChart Import

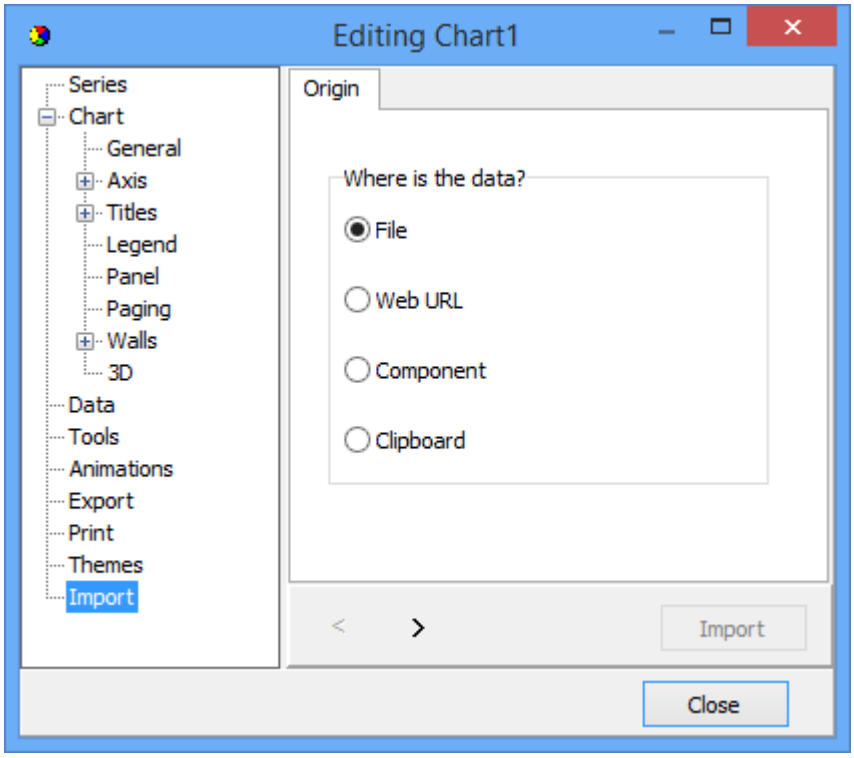
Rev. February 2015 http://www.steema.com

The TeeChart TDataImport class is included with current versions of TeeChart Pro VCL/FMX.

“Import” is a new item in the Chart editor dialog:



That enables importing data to a Chart from a large quantity of different sources:



This editor (which can be used both at design and runtime – not all of the options are currently visible in FMX), uses a component that can also be used at runtime to import data, for example:

```
var Import : TDataImport;

Import:= TDataImport.Create(Self);
Import.Chart := Chart1;

Import.Import(   xxxxx  );
```

"xxxxx" can be one of the following:

- TStrings (ie: Memo1.Lines, ListBox1.Items, etc)
- TStringStream
- TDataset (any kind, like SQL query, table, client dataset, memtable, etc).
- TDataSource
- String (simple text)

- Array of Double / Integer
- Another Chart
- Another Series (one or many)
- TStringGrid cells
- XML and IXMLDocument
- JSON and TJSONValue
- REST and TRESTResponse (XE5 and up)
- File (any kind of file supported)
- URL (any kind supported) http and https
- Excel (xls and xlsx) <--- Windows only, needs Excel installed
- TImage, TImageControl (png, jpeg, gif, bmp, etc)
- ClientDataset files (*.cds)
- TLabel and TEdit
- TColumn (FMX only, Grid columns)
- RTTI (arrays, records, properties, TList, TCollection, etc)
- OpenDocument (not yet implemented)
- Sensors (not yet implemented, needs some kind of auto-refresh)
- ZIP files (any supported files inside the zip)
- HTML <table>
- Dbase and Paradox tables (*.db and *.dbf) using BDE

Importing generates new Chart Series and fills them with the appropriate data.

The goal is to make the import process somewhat “smart”, detecting the data format and quantity and choosing the most suitable chart style.

For example, importing text (string or TStrings) will be analysed to determine if text format is CSV (Comma Separated Values), JSON, XML or HTML.

For CSV, the field separator will be automatically detected (comma, space, tab, semicolon, etc).

Examples:

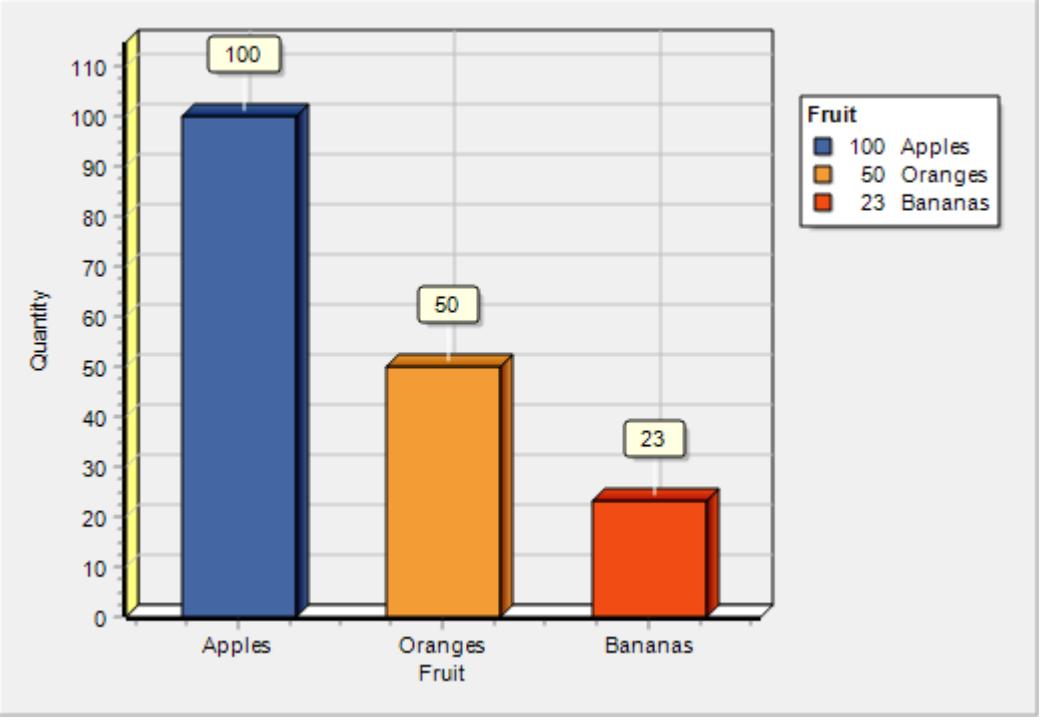
If you have this text on a "test.txt" file:

Fruit,Quantity
Apples,100
Oranges,50
Bananas,23

Calling:

```
DataImport1.Import('test.txt');
```

Generates this chart:

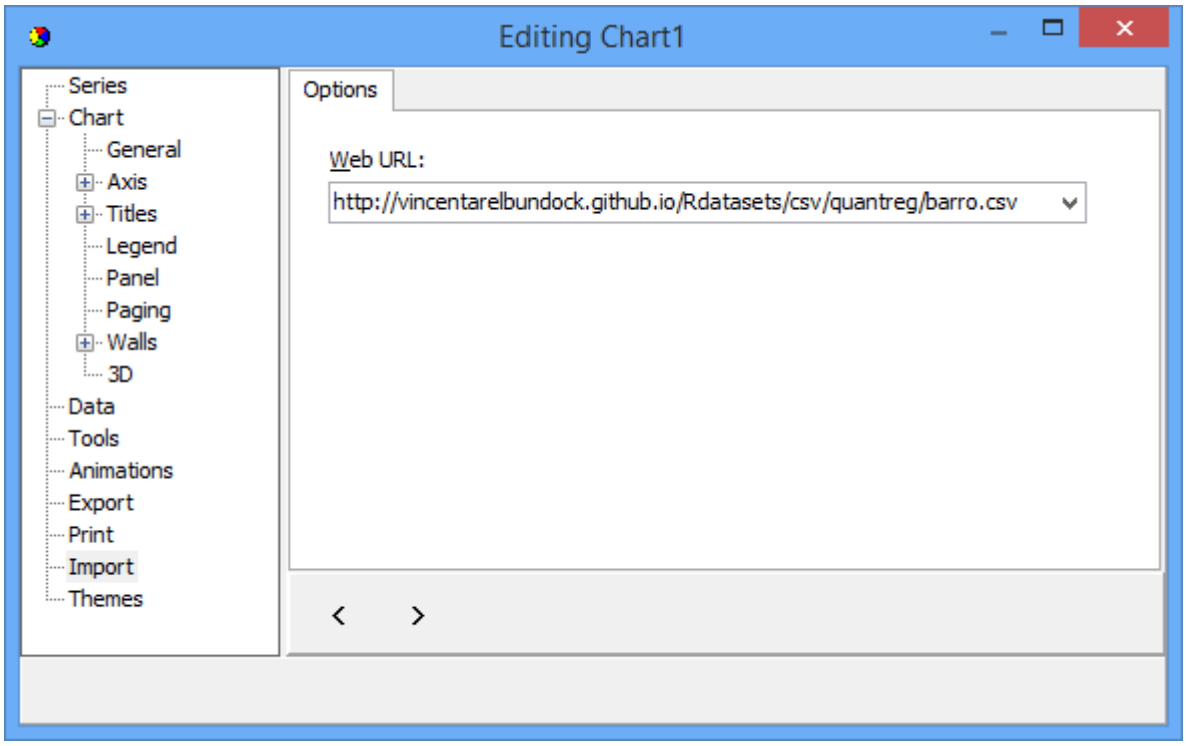


Identical results are obtained if text is inside a Memo control or any TStrings component:

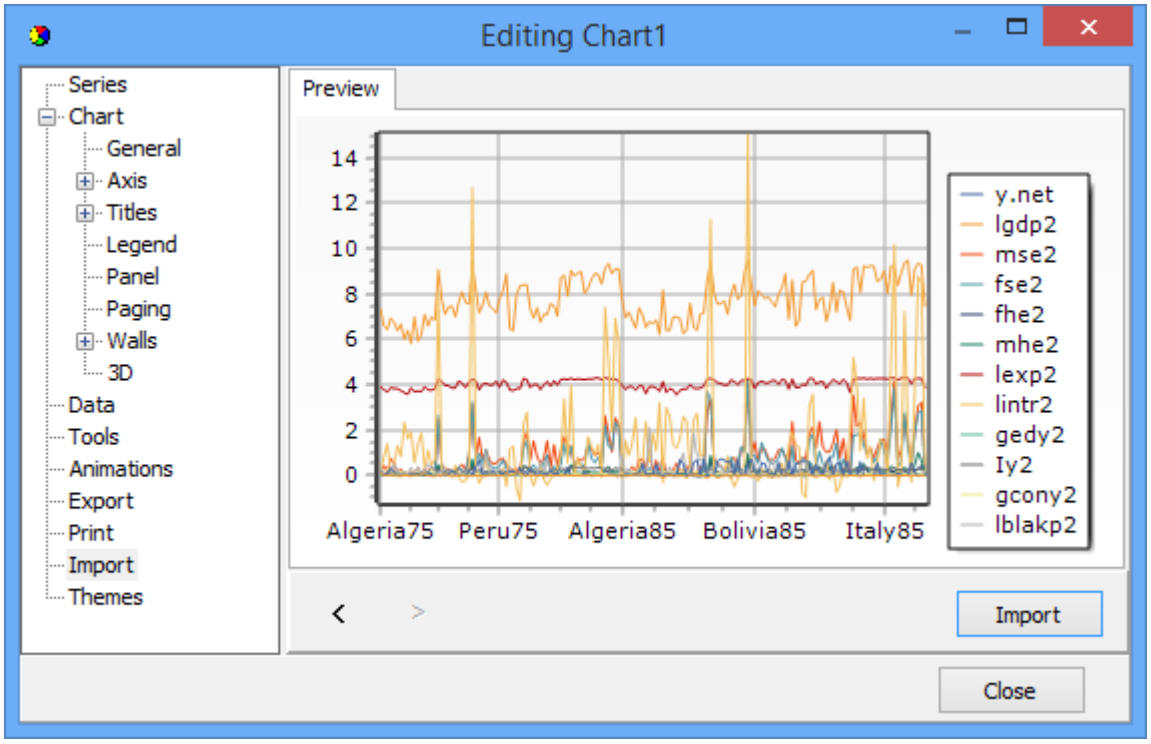
```
DataImport1.Import( Memo1 );
```

```
DataImport1.Import( MyStringList );
```

Multiple series (columns) are automatically imported, for example this CSV web data:



Generates this chart:



Database data works similarly, using Dataset fields as columns.

The first text (string) field is used for labels, and all numerical-capable fields are plotted.

XML, JSON (and REST) data formats have the special characteristic they can be hierarchical.
That is, any field value can be a subobject with more fields.

For simple XML and JSON structures, the first suitable “array” of items is automatically detected.

For example this JSON data:

```
[
  { "name": "Mike", "age": 38 },
  { "name": "Julia", "age": 17 },
  { "name": "Charles", "age": 23 }
]
```

or its XML equivalent:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Persons>
   <Person0>
      <name>Mike</name>
      <age>38</age>
   </Person0>
   <Person1>
      <name>Julia</name>
      <age>17</age>
   </Person1>
   <Person2>
      <name>Charles</name>
      <age>23</age>
   </Person2>
</Persons>
```

To use sub-object data, currently you should manually pass a “RootElement” (or “Path”):

```
DataImport1.Import( ‘data.json’, ‘Sales.Cars.Europe’ );
```

Calling Import by code using arrays

```
DataImport1.Import( [ 123, 789, 456 ] );

DataImport1.Import( MyArrayOfDouble );
```

RTTI

When importing data structures, fields and properties from classes and records will be examined using RTTI.

```
DataImport1.Import( TValue.From( xxx ));
```

“xxx” can be:

- Array of TClass
- Array of record
- Array of Float (Single, Double, Extended)
- Array of Ordinals (Byte, Word, Integer, Cardinal, Int64, etc)

- TList<TClass> (generic)
- TList<record> (generic)
- TStack and TQueue (generic)
- TCollection

Example:

```
type
  TPerson=class
  public
    Name : String;
    Birth : TDateTime;
    Height : Single;
  end;

var Persons : Array of TPerson;

…
DataImport1.Import( TValue.From( Persons ));
```

The output chart will show each TPerson “Height” and “Name”.

Automatic Formats

The import process can determine the charting style based on analyzing the data.

```
DataImport1.Style := isAutomatic;  // default
```

Cases:

- Normal output are 2D series (Line, Bar, Pie, etc). One column or field per series.
- 3D series (Surface, etc) are generated when there are 3 fields named “X”,”Y” and “Z”.
- Geographic output (World series) is emitted when data headers indicate a continent or country name.
- Financial charts (Candle series) are generated when fields are named “Open”, “Close”.

Titles

Chart titles will be automatically extracted from data.

This can be controlled with the Keep property, for example avoiding changing the chart Legend title:

```
DataImport1.Keep := [ksLegendTitle];
```

- Chart Title (the main top title)
- Legend Title
- Axes Title

Splitting Axes

When importing more than one field, data will be analyzed to set the appropriate axis to each series.

Similar series (that overlap their min max ranges) are assigned the same axis.

For example, you might import 4 series and get 2 associated to the chart Left axis, and the other 2 to chart Right axis, according to series values similarity.

Grouping fields

When importing more than one field, if data from one field is sequential, it will be considered as the “X” or “ID” value for the other fields.

Sample Data

The Import test example uses a folder filled with many files in different formats.