

Icons in Delphi: new possibilities with IconFontsImageList & SVGIconImageList

by Carlo Barazzetta * (carlo.barazzetta@ethea.it) - September 14, 2020

* author of IconFontsImageList and SVGIconImageList

Index of topics

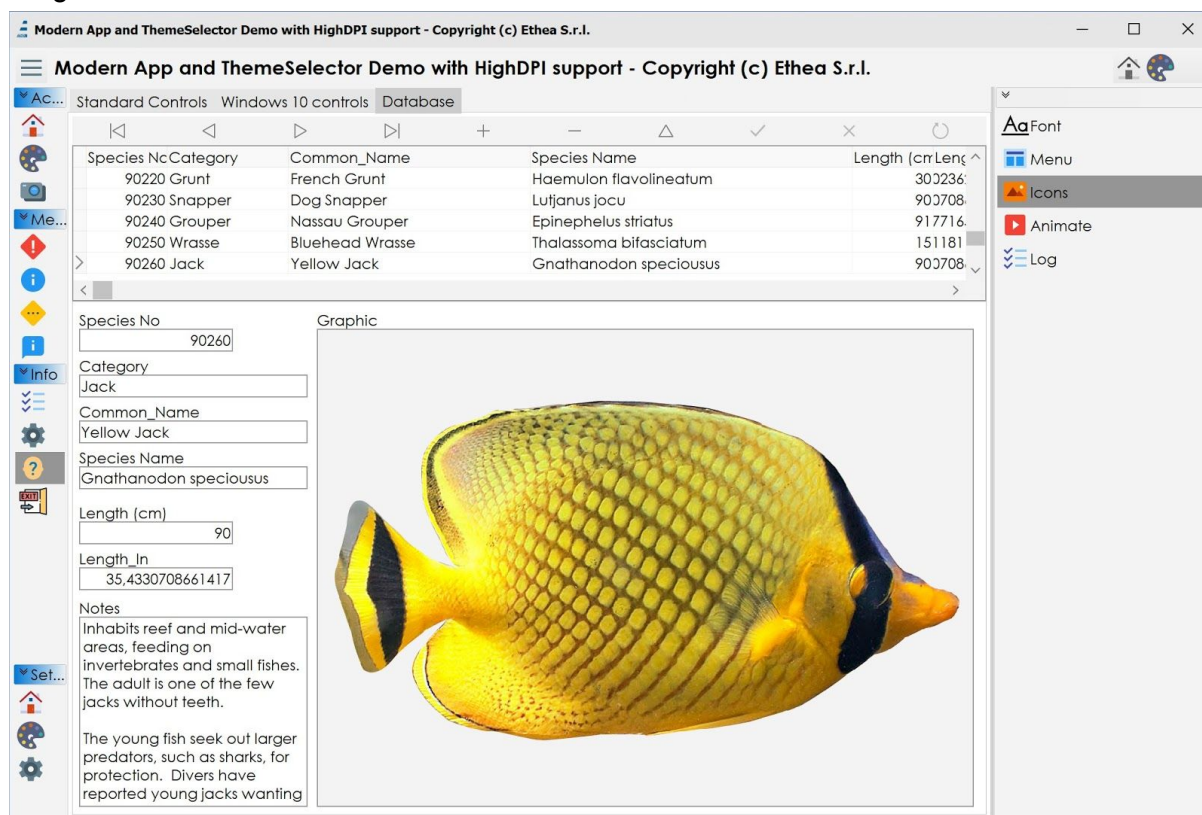
Foreword	2
A brief of history	3
Icons in Windows	3
Google - Material Design	3
Light and dark theme	4
The advent of High-DPI screens (2K, 4K, 8K)	4
Icons in Delphi and VCL	5
TImageList	5
Delphi 10.3: TImageCollection and TVirtualImageList	5
TImageCollection	5
TVirtualImageList	6
Advantages of pairing TImageCollection with TVirtualImageList	7
Limitations in the use of Bitmap images	8
The solution? Icon fonts or SVG	8
An must-have point: "scalability"	8
Icon Fonts	9
SVG Icons	9
Evolution of IconFontsImageList	9
Evolution of SVGIconImageList	10
Two rendering engines for SVG format	11
Embarcadero's Get-It (package manager) availability	12
IconFontsImageList and SVGImageList: features and instructions for use	13
Components	13
Collection + VirtualImageList for old Delphi versions	13
IconFontsImageList component editor (new version)	14
SVGIconImageList component editor (new version)	15
Native VirtualImageList support	16
Utilities	17
IconFonts CharMap and Icons catalog	17
SVG Icons Explorer	18
Demo projects and documentation	18
Thanks	19

Foreword

About eleven months ago, I started the IconFontImageList project (SVGIconImageList followed soon). The reason was I have been in the need of some additional features to modernize existing applications (both VCL and Firemonkey).

I decided to provide them for free because I felt there was a widespread need of better support in using icons within Delphi. Some third-party components are available, though. My projects became very popular in a few months and this proves I was right thinking the community could benefit from my work!

As a proceeding, I decided also to write this article to explain the circumstances and reasons leading to their birth and how they grew to their maturity. Today, I consider them a must-have when it's up to development of modern applications, as you can see in this image:



After an initial part of this document, that acts as an introduction to the topic and serves to highlight some of the limits tied to the use of native components, we'll delve into genesis and evolution of my components, up to the current version.

One of the main topics is the transition from use of a classic TImageList (VCL) to a pair of TImageCollection and TVirtualImageList (introduced in Delphi 10.3). This is the prerequisite to enable a proper High-DPI support while using icons in modern applications.

Nonetheless, this article is also valid for those using Delphi versions prior to 10.3. An equivalent (or similar) support has been implemented and is available while working with older versions (sometimes adding compatibility down to Delphi 7!).

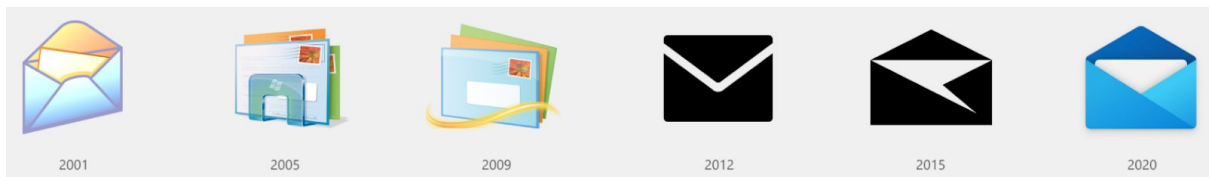
The version for Firemonkey will also be discussed in a future article...

A brief of history

To introduce the context, it is useful to take a step back and observe how the use of icons in the computer world evolved. We will focus on the Windows environment.

Icons in Windows

Without delving too much into the evolution of icons in the computer world (if you interested in a global discussion about it, check out: <https://historyoficons.com/>), we are going to focus on the Windows platform. We were used to very colorful and real-world-lookalike icons, until 2009. The revolution arrived with Windows 8 and therefore continued with Windows 10.



Suddenly 2 facts happened: icons became flat and extremely stylized, up to the choice of monochrome. In later versions (Windows 10) icons have changed again: still very stylized, basically with a primary color and some shades reintroducing a bit of three-dimensionality:

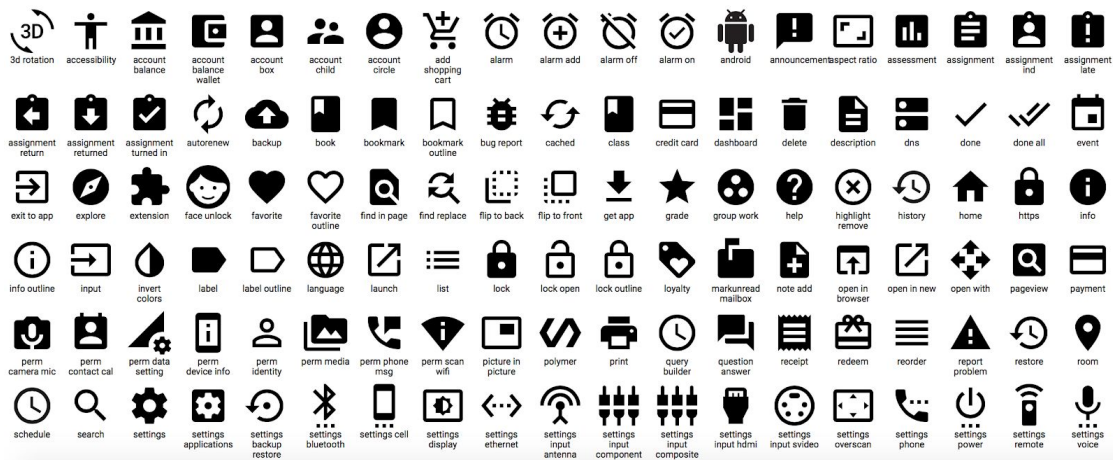


Google - Material Design

The Web has also seen its revolution. The main driver has been Google guidelines for the "Material Design" ecosystem. A huge success of this approach also led to a sort of convergence about the meaning of each icon. They are now identified with a name and a category, and you can find the same icons used for the same purpose across different applications (in the following figure, an example with material icons for "action" category).

Material Icons

action



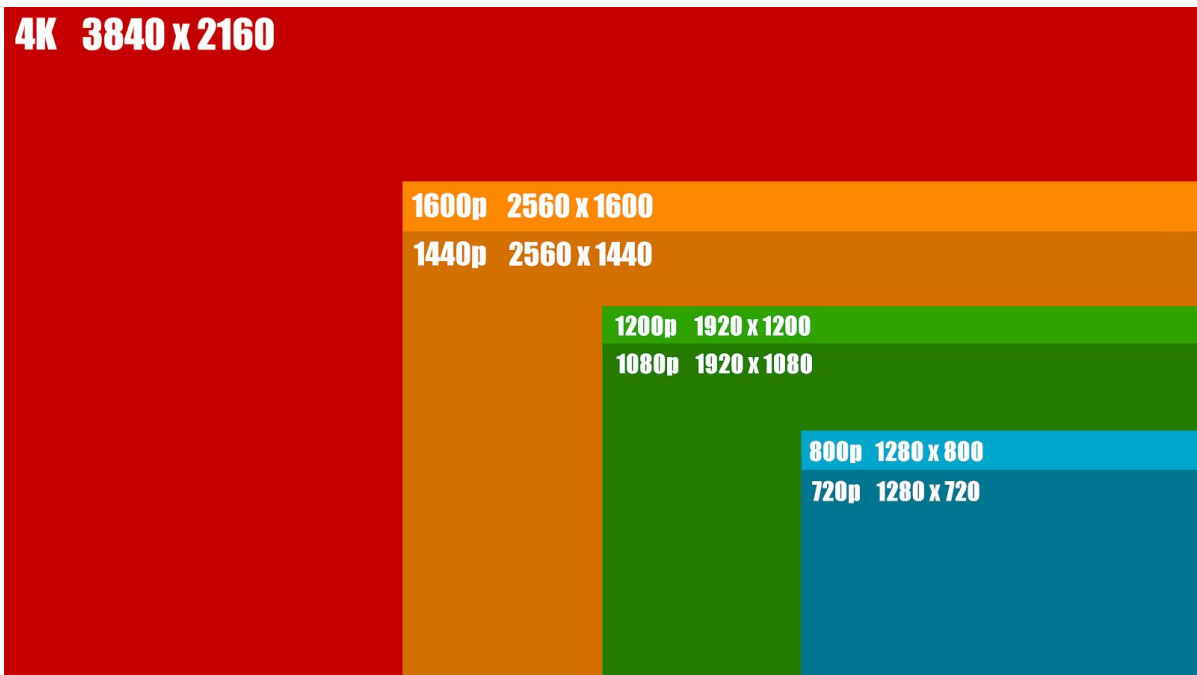
Light and dark theme

Another small revolution, still very recent (late 2018), involving almost all most popular applications has been the introduction of the "dark" theme. Initially, it has been depicted as a convenient choice to save some battery life of mobile phones (Windows Phone 7), then it proved to be very appreciated, especially by younger people, and it quickly became the default option on desktop too.

The advent of High-DPI screens (2K, 4K, 8K)

Lastly, more and more notebooks are now equipped with 2K resolution displays and more and the new standard for professional use is 4K (the 8K frontier is just around the corner). With higher resolution screens, the final user needs to adjust the font size or, more precisely, on the DPI (Dots Per Inch) factor multiplier. A significant change with respect to the past, where this parameter has been untouched for decades (in Delphi it has been called PixelsPerInch).

In modern applications, to achieve optimal rendering, application icons must have the ability to adapt (scale) to any DPI. With a 4K resolution there are 5 user selectable DPI factor multipliers (100%, 125%, 150%, 175%, 200%).



Starting with the latest versions of Delphi, High-DPI support has been added by Embarcadero: once enabled, your application needs to properly manage its appearance, in order to adapt to each different scaling factor. While components, according to the Delphi version you are using, have increased their ability to automatically adapt to different scale factors, management of Icons is still difficult and in many ways it represents a problematic aspect: let's see why.

Icons in Delphi and VCL

Let's start with what Delphi supports out-of-the-box and the “Virtual” evolution introduced in the 10.3 version. We'll dig into some significant advantages and some limitations of native components.

TImageList

Historically, the component with the responsibility to manage icons in a Delphi application has been **TImageList**, a collection of **bitmap** images with a fixed size provided in various image formats (BMP, GIF or PNG, according to correspondent support available in Delphi versions). The TImageList acts as a provider for other GUI components (TAction, TToolBar, TButton, etc ...). Icons are referenced through a numerical index identifying the specific icon. Another aspect we saw evolving over the years has been management of transparency. From the famous “pixel in the lower left of the image” to the graphic formats (GIF, PNG...) where transparency has been coded directly in the image data. With the arrival of GDI+ support (starting with Delphi XE4) there has been an evolution in quality and performance over this aspect.

Delphi 10.3: TImageCollection and TVirtualImageList

A high DPI aware application should be able to provide icons fitting several different DPI factors. Therefore it is important to separate the concept of “size” from the concept of “appearance”. Starting with Delphi 10.3, two new components are available to be used instead of TImageList: **TImageCollection** and **TVirtualImageList**.

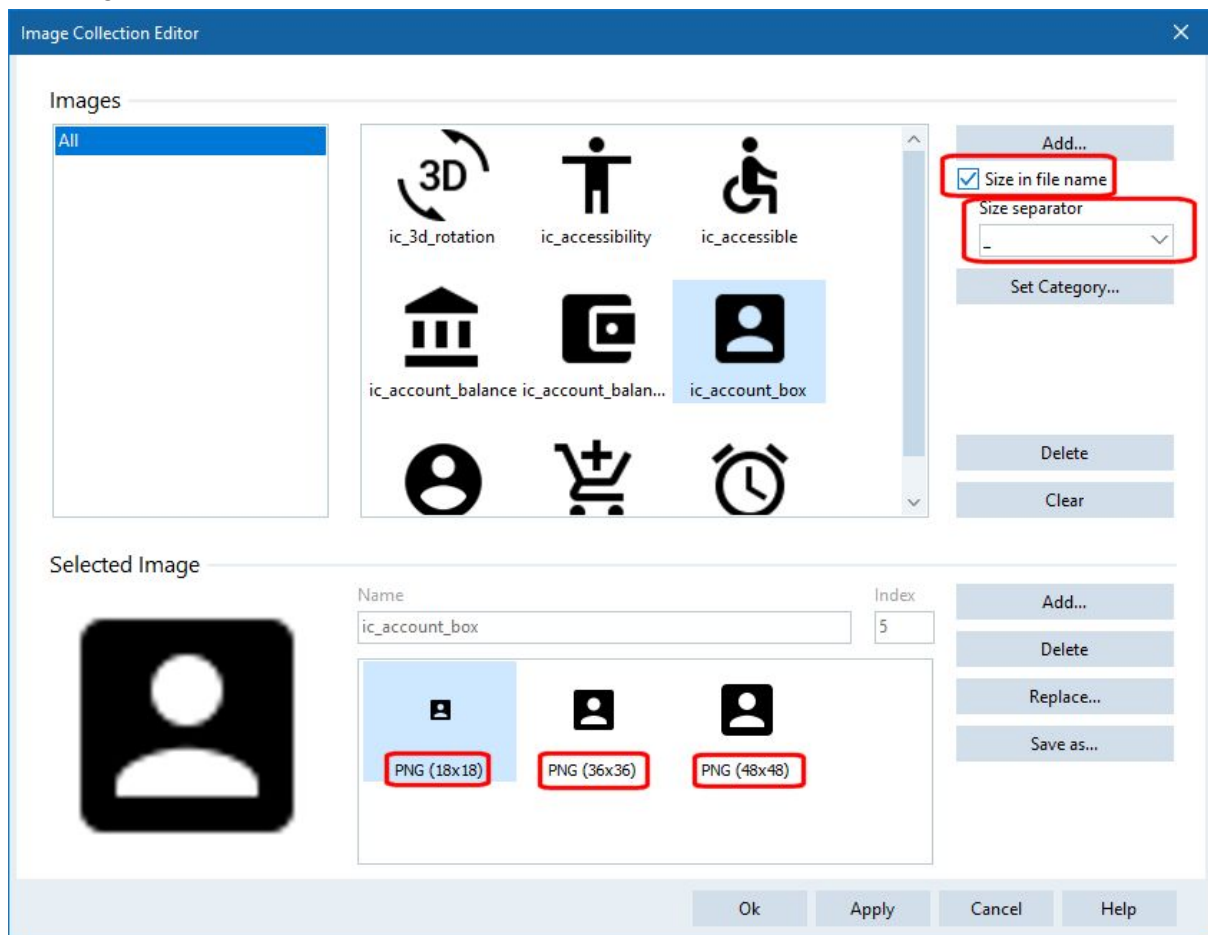
TImageCollection

This component collects a list of "Bitmap" images. It is possible to load them from files (of many formats and sizes) in order to populate the collection

Usually, the component is put onto a TDatamodule in order to be available for use by all the application forms needing it.

One of the advantages (not always known to everyone) is the ability to add the same icon in the collection with multiple (different) resolutions: it is necessary to pay attention and set the "Size in filename" and "Size separator" options, before loading the icons through the “Add” button, within the component editor.

For example, you can set the underscore character as "size separator" and load three files: ic_account_circle_18pt.png, ic_account_circle_36pt.png, ic_account_circle_48pt.png, the collection will show a single icon named ic_account_circle but it will contain three versions of the image, with three different resolutions: 18, 36 and 48 pixels.



Please consider that having different resolutions is only the starting point in the quest for scalability. In fact, the component also contains the scaling algorithm defined in the

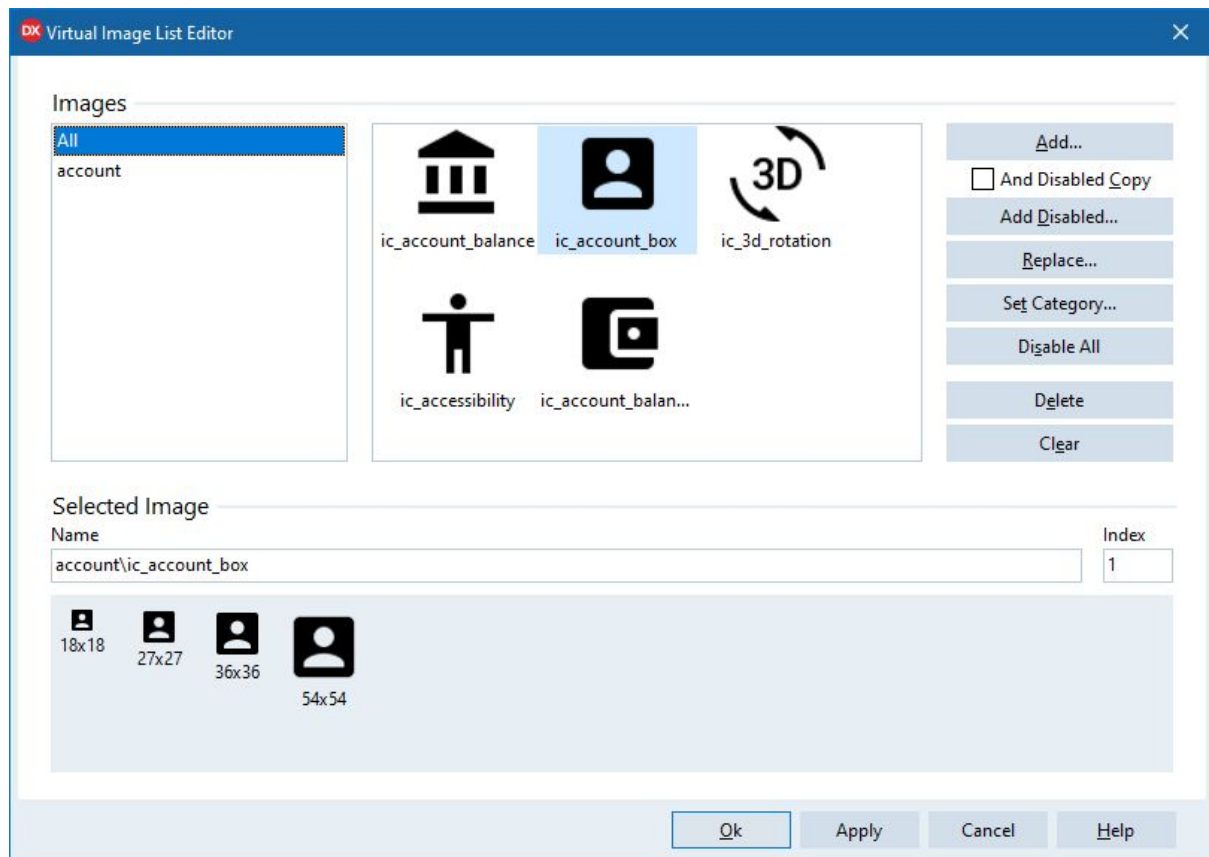
“Interpolation mode” property which by default is set to `icIMModeHighQualityCubic` value: according to the chosen algorithm, quality is preferred over performance or vice versa. Another important fact is that icons are identified by name, with the possibility to also assign a Category. This will come handy later, to filter entries while using a `VirtualImageList` component.

TVirtualImageList

In order to make use of icons contained in a `TImageCollection`, you need a `TVirtualImageList` component. You can place it on each form and make it reference the Collection.

The `TVirtualImageList` component states the size (Width and Height) of the icons to be included, but it is important to keep in mind that this size is always to be considered with respect to a screen with 96 DPI.

The advantage of placing a `VirtualImageList` on each form also consists that you can choose “only” the images you need in that specific context: in the following example you can see we added only 5 of the images in the collection. You can even sort them in different positions, because they are referenced by name (and not by position). You can also set a category for the icons: in fact the icon will be called “category \ name” (eg “account \ ic_account_box”) even if, unfortunately, the categories defined in the `TImageCollection` are not automatically fetched while adding images to the `VirtualImageList`.



In the lower part of the component editor (check the above screenshot) a preview of the icon is provided, exposing eventual different “scale factors”: since our `VirtualImageList` has been configured to have dimensions 18x18, the starting icon is shown together with other three versions: 150% (27) , 200% (36) and 300% (54).

It is also possible to provide icons for the “disabled” state. At component level, you can specify the suffix (`_disabled`) and the corresponding opacity value (default 125) to provide the “disabled” version of the icons.

Advantages of pairing TImageCollection with TVirtualImageList

The advantages of this setup are important and surely it is highly recommended to switch from the old TImageList approach to this new one.

The main pros are the following:

- 1) each form can host several TVirtualImageList with different sizes (18x18 for toolbar icons and 24x24 for button icons), sharing the same collection;
- 2) each TVirtualImageList references the collection entry by name and not by index, allowing you to add icons to the collection even in a second time and letting you free to even change their position in the list: your GUI elements then will reference images in the TVirtualImageList by "ImageIndex" (starting from Delphi 10.4, it is possible also to reference them by "name");
- 3) it is possible to add in the TVirtualImageList only the icons you actually need in that specific form, without having to "generate" the entire set of icons provided in the collection. In fact it is the TVirtualImageList that is aware of the actual DPI value of the monitor the form belongs to. Therefore the component can "query" the collection to "draw" the icon with the required size but with the scale factor corresponding to the monitor!

In addition to the other advantages described so far, you should keep in mind that this architecture is the only one suitable for a truly High-DPI aware application. Whenever we have more than one form in the application, it is possible different forms could be dragged across different monitors, potentially with different DPI factor multipliers. In this case we want icons of that specific form to be resized and not the whole original collection.

Limitations in the use of Bitmap images

The major limitation of TImageCollection is a consequence it uses bitmap images of "fixed" size. This brings a number of consequences:

- 1) you need to provide icons at different resolutions and to load them all into the collection. This causes useless occupation of memory and resources (all bitmaps in all resolutions) with the possibility most of them will never be used in the life cycle of the application;
- 2) you rely on "rescaling" algorithms when an icon size not matching those available in the collection is required;
- 3) you need to double the effort in order to provide the application with a light / dark theme mechanism. Furthermore, you may even need to provide several set of icons (of different color schemes) in order to match the actual VCL style selected by the user.

The solution? Icon fonts or SVG

To modernize an existent Delphi (VCL) application we must necessarily address and solve the following issues:

- provide the application with a set of modern icons;
- provide the application with a set of icons capable to properly scale and adapt to screens with different DPI values;
- support the ability to adapt the icon set to the color scheme (light / dark theme) selected by the user.

Another relevant issue, in my case, has been the need to find a solution compatible with older Delphi versions (prior to 10.3), lacking TImageCollection and TVirtualImageList components.

I was aware that a solution compatible with previous versions would have been a plus. Many Delphi developers are in the need to “modernize” existing applications, at least from a visual point of view, without being forced to upgrade to a different Delphi version.

An must-have point: "scalability"

The main issue to be addressed was to achieve scalability of icons. The use of vector graphics technology, instead of classic interpolation algorithms over bitmaps, enables us to reach the goal. Considering we have already stated the need of having “stylized” icons, vector graphics definitely represent an optimal solution.

Looking at the web world, where designers are used to pay a lot of attention over image quality, we can find similar issues and needs. The tentative answer is composed of two kind technologies, somehow related each other:

- 1) Icon Fonts
- 2) SVG Icons

Icon Fonts

The use of the Web -Fonts or Icon Fonts is currently gaining a lot of popularity. These days, there are dozens of rich and popular collections, we can mention a couple:

<https://github.com/Templarian/MaterialDesign-Font> (almost 5,600 free “Material-Design” icons)

<https://fontawesome.com/icons?d=gallery> (almost 8,000 icons in total, 1,600 free icons)

To use an Icon Font offers many advantages:

- 1) the collection is already complete (all icons show a consistent appearance);
- 2) no need to distribute or upload hundreds (thousands) of single png files; a single (or a bunch of) Font Files contain all the icons;
- 3) perfect scalability, thanks to use of vector graphics;
- 4) it is possible to change the color attribute of the icon.

The last point, actually, also represents one of the biggest limitations while using Icon Fonts: it is not possible to use multiple colors within the same icon.

SVG Icons

Scalable Vector Graphics (SVG) is a technology capable of defining “vector” images. These are perfectly scalable. It was introduced back in 2001 and for many years it remained a technology relegated to a limited number of use cases, with little uses for icons. Mainly this was due to the requirement of a rendering “engine”: nowadays, thanks to current CPU / GPU performances, it is no longer an issue to “draw” icons on the fly.

The use of SVG Icons also brings other advantages:

- 1) there are many collections of SVG icons out there;
- 2) you don't need many files (matching each resolution) but just a single SVG text file;
- 3) perfect scalability by design;
- 4) you can easily manipulate XML elements of the SVG text to change attributes, such as color.

The only disadvantage is the need to have an "engine" capable of supporting all the features of the SVG standard. We will see how this problem can also be solved.

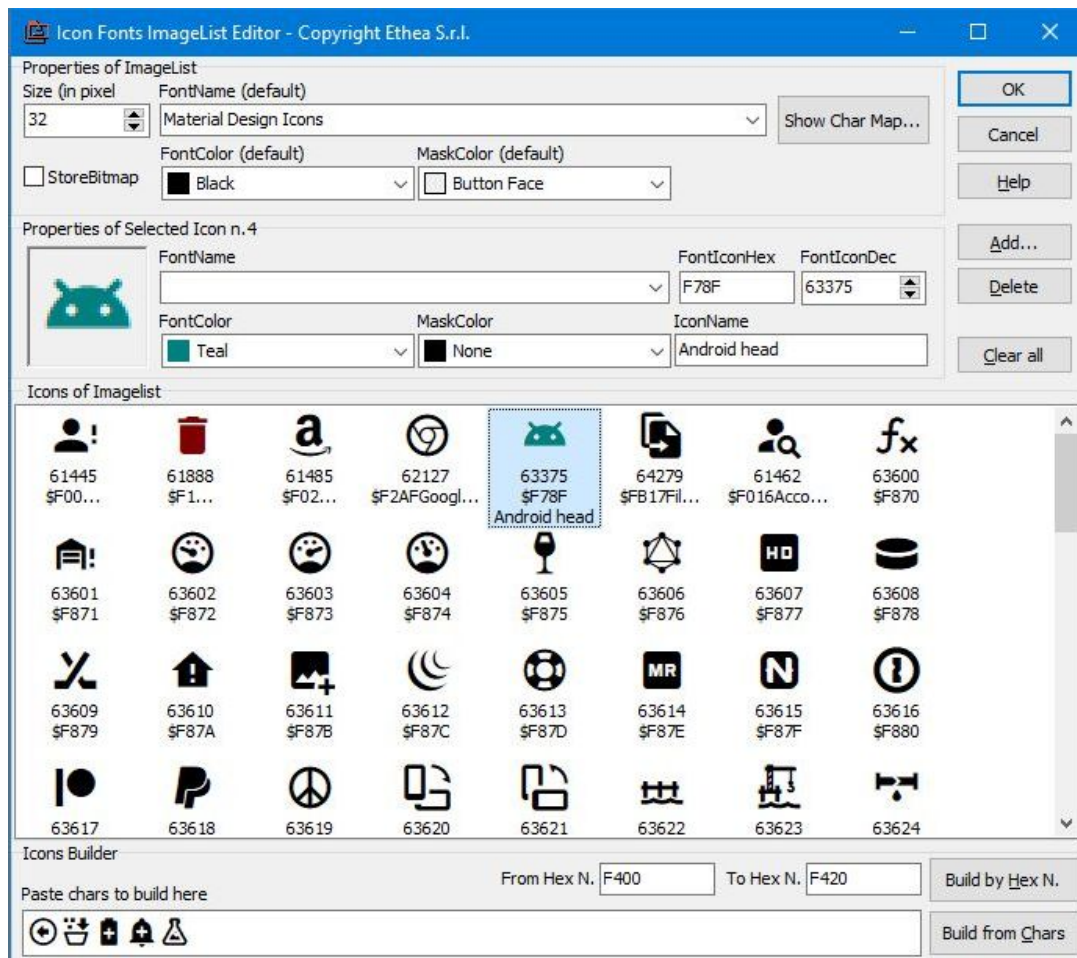
I just had to find a way to apply all these technologies to the Delphi world ...

Evolution of IconFontsImageList

Development started in November 2019: idea was simple, inspired by a project by Luca Minuti (<https://github.com/lminuti/FontIconEditor>). In this project we can see how to obtain an icon from an icon font and add it to an ImageList. My solution actually does exactly the opposite: replace a classic ImageList with an ImageList of dynamically generated icons. The source would be one or more icon fonts and thus able to "scale" and change color easily.

The first Open-Source release on GitHub of **IconFontsImageList**, on November 25th, has been a success: starting from the first released version, the component allowed to generate icons from different fonts, conveniently applying a primary color together with a background one. A Demo project was included to illustrate how easy it was to obtain icons of any size, perfectly capable to scale and "adapt" to changes in style (light or dark) of an application.

The component is provided with an advanced editor that can also capture icons from the character map of the font. (in the following figure the "old" component editor).



In the following weeks there has been many releases, with new features:

- January 30th: provided VCL support back to Delphi 7;
- March 25th: beta version for Firemonkey comes to light;
- March 28th: support extended to fonts that use "surrogate pair" Unicode characters;

- April 13th: custom Charmap has been added, to display "surrogate pair" characters Windows charmap does not support;
- May 7th: component editor added to the FMX version;
- May 27th: metadata of "Material Design" and "Font Awesome" icon fonts are added with the ability to access CharMap icons by name;
- June 19th: version 2.0 adds GDI+ support for rendering and a new component *IconFontsImage* to display an icon inside an image.

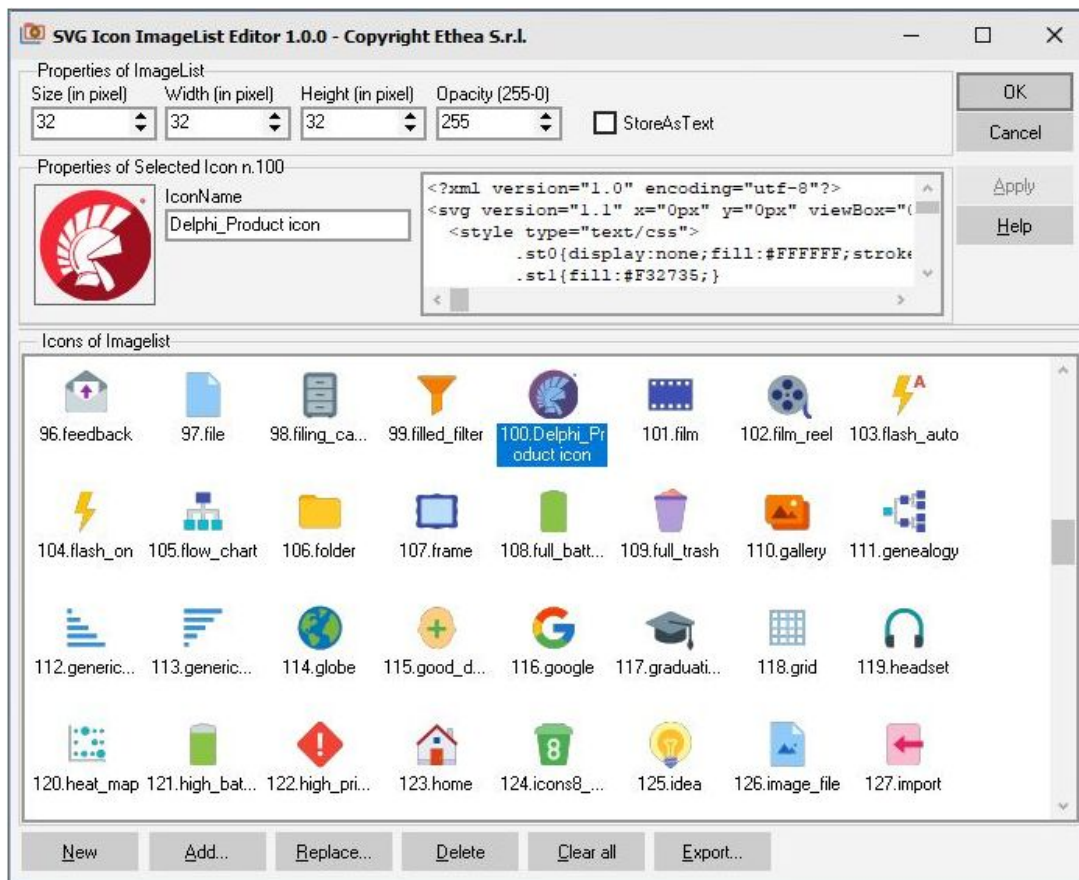
In August, **a complete refactoring of the library has been done** and version 2.2 has been released on August 27th. This release contains the transformation into "Collection" + "Virtual": we will cover details in the section dedicated to the library internals.

Evolution of SVGIconImageList

Along with the development of *IconFontsImageList*, I've received requests (from many different sources) about the ability to have coloured icons instead of just monochrome. I got suggestions to build a SVG based component but I had to face the problem that no native SVG engine was available for (free) use in Delphi. Letting apart available commercial libraries, I recalled an old library, seen years ago, by Martin Walter, currently released for free (<https://development.mwcs.de/svgimage.html>).

After doing some research, I also found a project on GitHub, showing no recent activity (<https://github.com/ekot1/DelphiSVG>) but that included the library I was looking for. It also included an experimental *ImageList* (but with poor performance), as well as a drafted FMX support. Thus, all needed elements were available and I started to build the *IconFontsImageList* "twin". The first version was released on May 24th.

The structure and component editor of **SVGIconImageList** are very similar to *IconFontsImageList*: this time images are defined through SVG text and they can be loaded from files on disk (in the figure the "old" component editor).



Even if **TSVG** library by Martin Walter has some rendering bugs, the first version of my component collected a lot of consensus and, again, I made some progress over time:

- May 26th: FMX version (Windows Platform only) is made available, component-editor included;
- May 28th: achieved backward compatibility up to XE6;
- June 5th: added TSVGIconImage component, to show an icon directly in an image component (Stretch / Proportional support included).
- June 9th: GrayScale and FixedColor properties are added, showcasing how flexible the SVG manipulation can be, enabling the developer therefore to adapt the icons to actual theme of the application;
- July 21st: **SVGExplorer** tool is provided, to navigate and preview SVG file icons on the Windows filesystem (as Windows is not yet able to do this natively).

Interest grows to the point of attracting other developers as collaborators: after a “frantic” month of 6-handed development (throughout August), version 2.0 was released.

Two rendering engines for SVG format

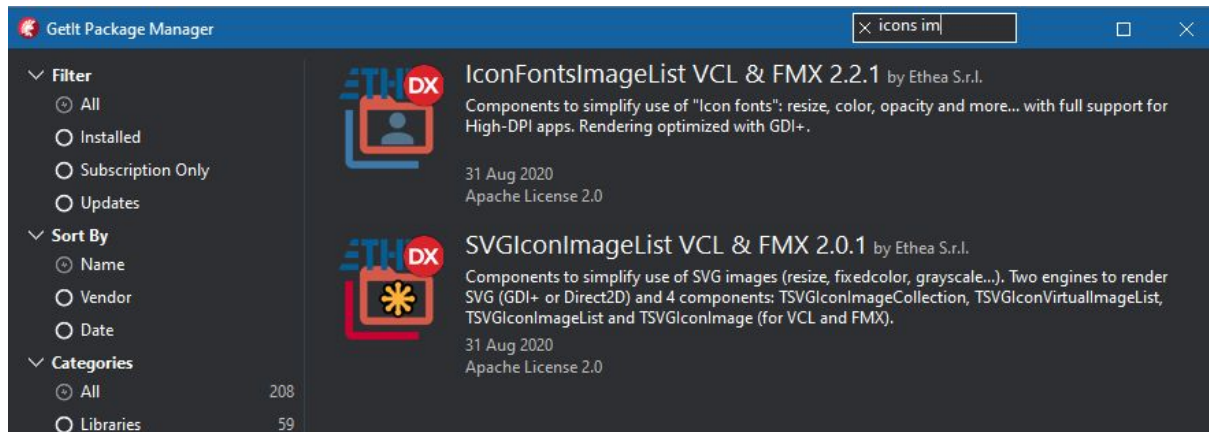
A big news about TSVGIconImageList library, version 2.0, is the added support for two different rendering engines:

- 1) native engine using Direct2D (provided by Windows 10), suggested option wherever OS provides support;
- 2) TSVG engine, by Martin Walter using GDI+ (revised, corrected and greatly improved version of the original project).

Beware, Windows 10 native support does not fully cover SVG functionalities, hence some icons may not be rendered correctly. In this case it is possible to switch to TSVG engine, by changing a simple compilation directive.

Embarcadero's Get-It (package manager) availability

The "consecration" of the libraries takes place with the publication on Get-It on September 2nd:







IconFontsImageList and SVGImageList: features and instructions for use

In this section we are going to illustrate some features of these libraries with respect to development of High-DPI enabled applications.

Given they are very similar and show specularly in several aspects, they will be treated together whenever this makes sense, stating differences explicitly where needed.

Components

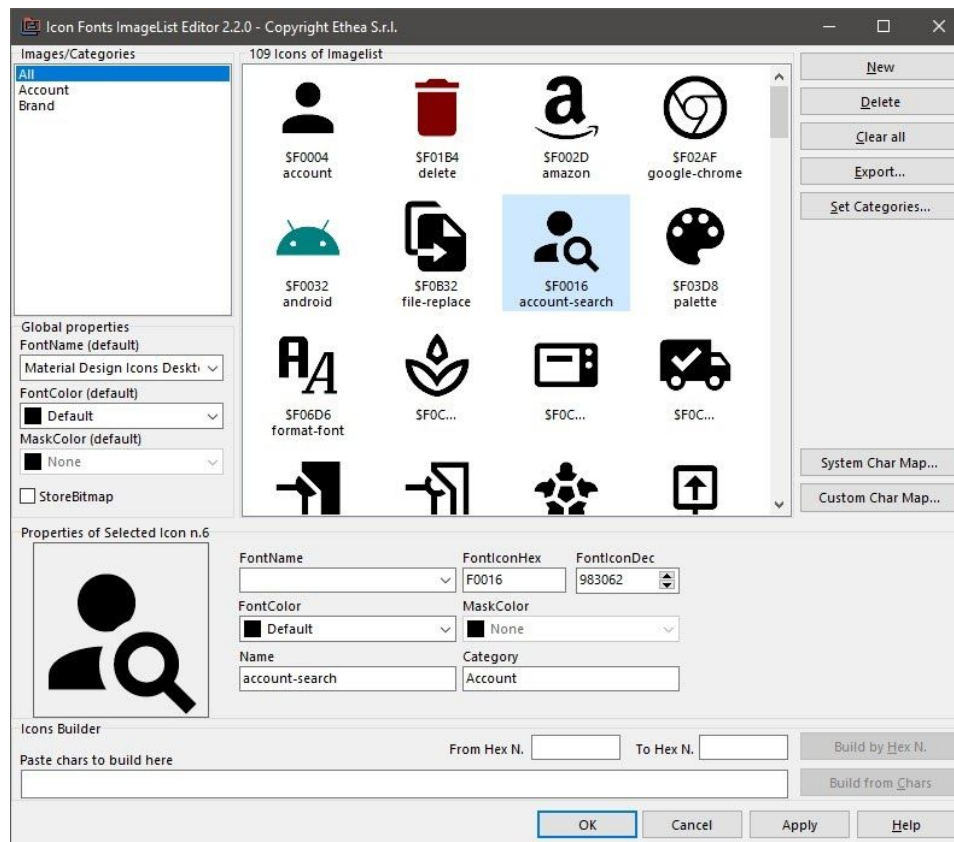
The following table summarizes the components of the two libraries, grouped as “twins”: in both libraries there are four components in total and they work in a similar way:

	TIconFontsImageList TSVGIconImageList	These are the “old approach” components, inherited from TCustomImageList, with the collection of Icons inside.
	TIconFontImage TSVGIconImage	These are “graphic” components to render a single icon on an image surface, scalable to any size.
	TIconFontsImageCollection TSVGIconImageCollection	These are the "new approach" components, acting as containers of scalable icon collections.
	TIconFontsVirtualImageList TSVGIconVirtualImageList	These are the "new approach" components to reference the collection and generate icons of a chosen size.

Collection + VirtualImageList for old Delphi versions

As already explained in a previous section, embracing development of High-DPI applications is not possible without embracing the Collection + VirtualImageList approach. Since these are not available in Delphi versions prior to 10.3, after providing a first "classic" version, a complete refactoring was done to make these components available starting from Delphi 7 (IconFontsImageList) and Delphi XE4 (SVGIconImageList). A redesign of the component editors also took place to make them more similar to the official component editor of TImageCollection and TVirtualImagelist.

IconFontsImageList component editor (new version)



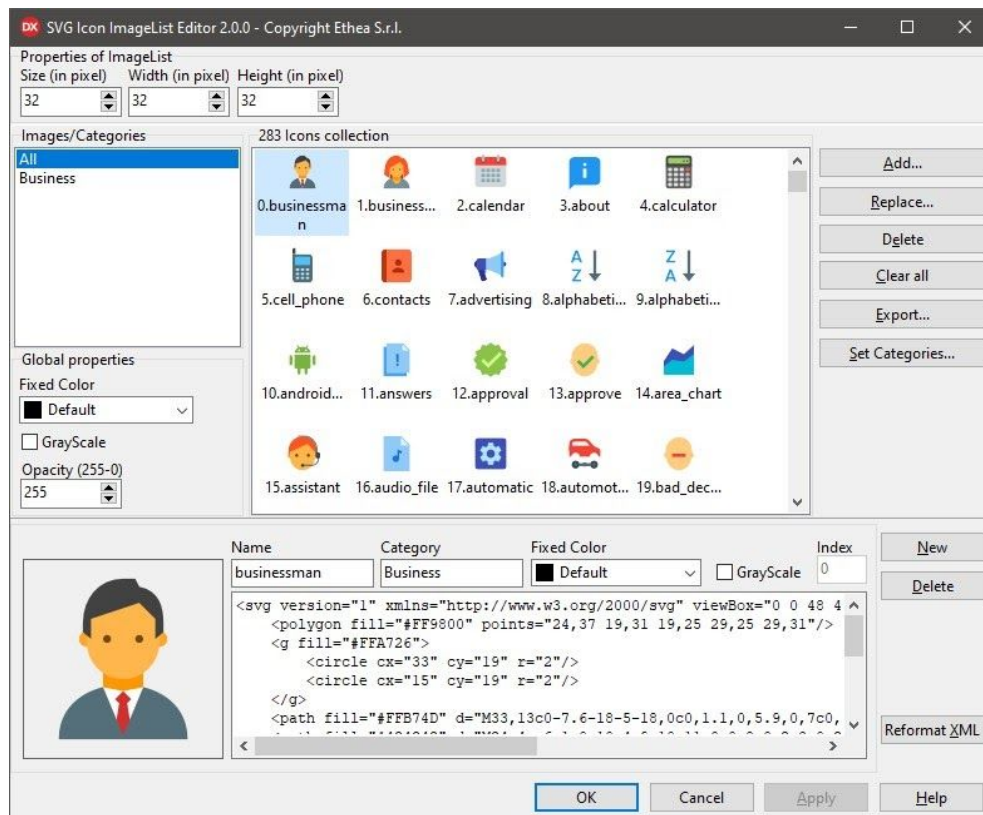
The new version of the component editor can be activated both from the "old" TIconFontsImageList and from TIconFontsVirtualImageList. It is also available using TIconFontsImageCollection component: in this case, you will not see the top panel, containing settings about size of the icons (check the above figure).

The component editor enables you to set some global values:

- **Size** (or **Width** and **Height**, when they differ) if it has been activated from the ImageList;
- **Default FontName**, to be used for all icons;
- **FontColor** (**MaskColor** is only available while using older Delphi versions, prior to XE4, lacking GDI+ support and relying on use of masked-bitmaps);
- **"New"**: creates a new entry in the list; you can then specify FontIconHex (or FontIconDec) to make it represent the corresponding Unicode "character" for the icon;
- **"Delete"** and **"Clear All"** respectively to delete the current icon or to clear the collection;
- **"Set Categories"** allows you to assign a category for the selected icons: clicking in the "Images / Categories" box filters the icons by category;
- For each icon it is also possible to:
 - specify an alternative **FontName** (if you are using icons from different fonts);
 - specify an alternative **FontColor** (and **MaskColor**);
 - set a **Name** and **Category**.

In the bottom part of the component editor window, the **Icons Builder** panel is always available, adding support to "import" icons from the Font. Just input the range of hexadecimal values or use the **"Paste"** field from the CharMap.

SVGIconImageList component editor (new version)



New version of the component editor is available for the “old” TSVGIconImageList, for TSVGIconVirtualImageList and also for TSVGIconImageCollection. In the latest case, you will not see the upper panel containing controls about size of the icons (visible in the above figure).

Through component editor you can set some global values:

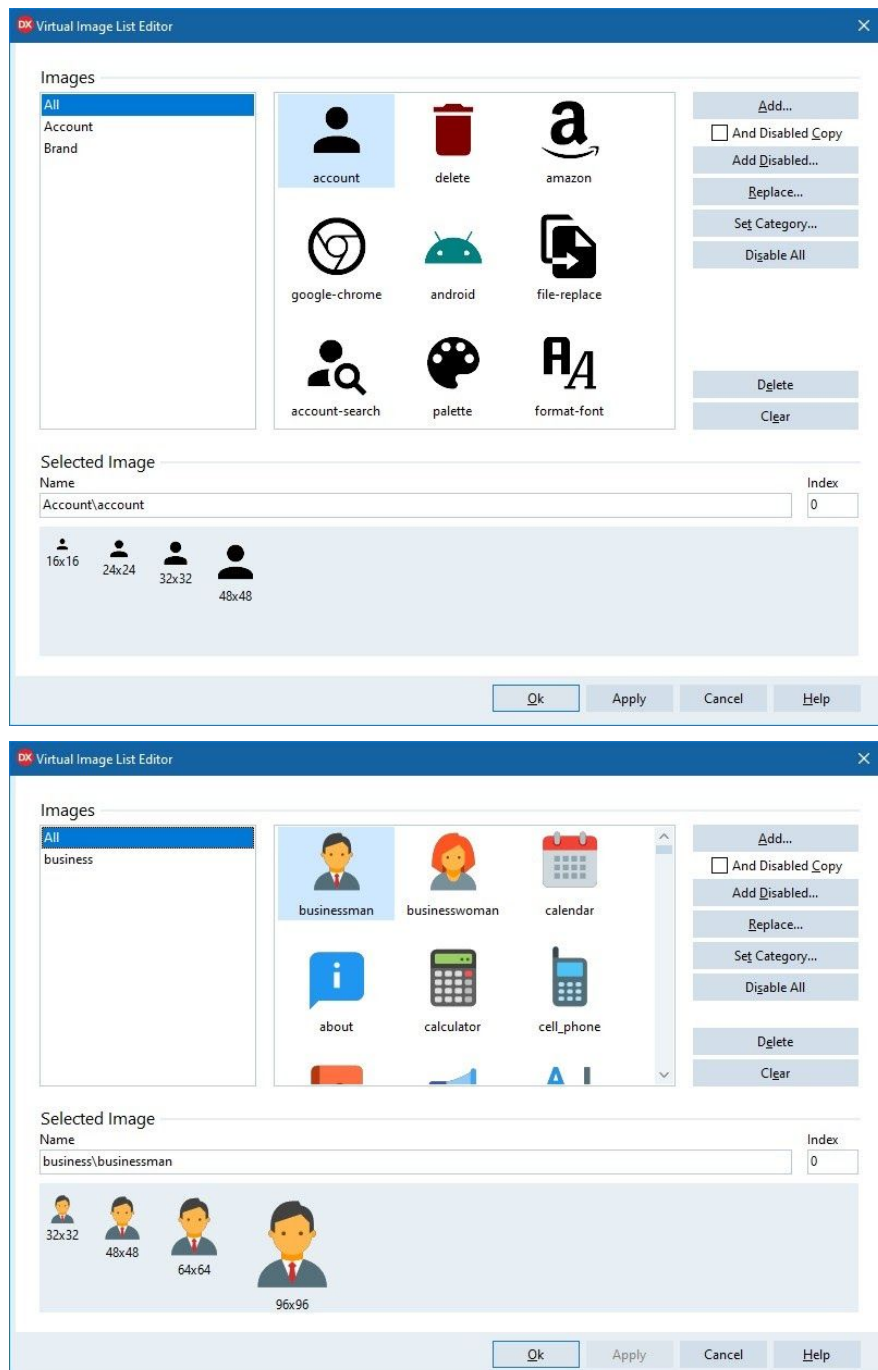
- **Size** (or **Width** and **Height**, whether they differ) when activated from an ImageList;
- **FixedColor**: if you want to “force” a single color to all icons;
- **GrayScale**: if you want to make all icons gray;
- **Opacity**: if you want to apply a transparency effect globally;
- **“Add ...”**: use this button to load external SVG files: corresponding icons will be created and the “name” guessed by the file name;
- **“Replace...”**: use this button to replace selected icons with others from files;
- **“Delete”** and **“Clear All”**: use these buttons respectively to delete selected icons or to clear the whole collection;
- **“Export ...”**: use this button to export entries to many SVG files (to the filesystem);
- **“Set Categories ...”** allows you to assign a category to selected icons: clicking in the “Images / Categories” box filters icons by category;
- For each icon it is also possible to:
 - define a **Name** and **Category**;
 - specify an alternative **FixedColor**;
 - apply **GrayScale** selectively to that entry;
 - edit (manually) the SVG text (through a small editor);
 - reformat XML text.

Native VirtualImageList support

As previously said, with Delphi versions prior to 10.3 it is possible to define a collection and use `IconFontsVirtualImageList` or `SVGIconVirtualImageList` components.

On the other hand, if you have a recent version of Delphi, it is suggested to use the native `TVirtualImageList` component, offering more the one surrogated one we added. For example, it has the ability to collect only icons actually needed by the specific form (as already discussed). It also implements the "PreserveItems" property, enabling you to preserve the icon index, even when the collection changes.

For these reasons, `IconFontsImageCollection` and `SVGIconImageCollection` inherit from `TCustomImageCollection` (when compiled with Delphi versions starting from 10.3). As a consequence, they can also be used in conjunction with `TVirtualImageList`: this way, the native component editor allows the developer to select icons provided by the collection, as shown in these figures:

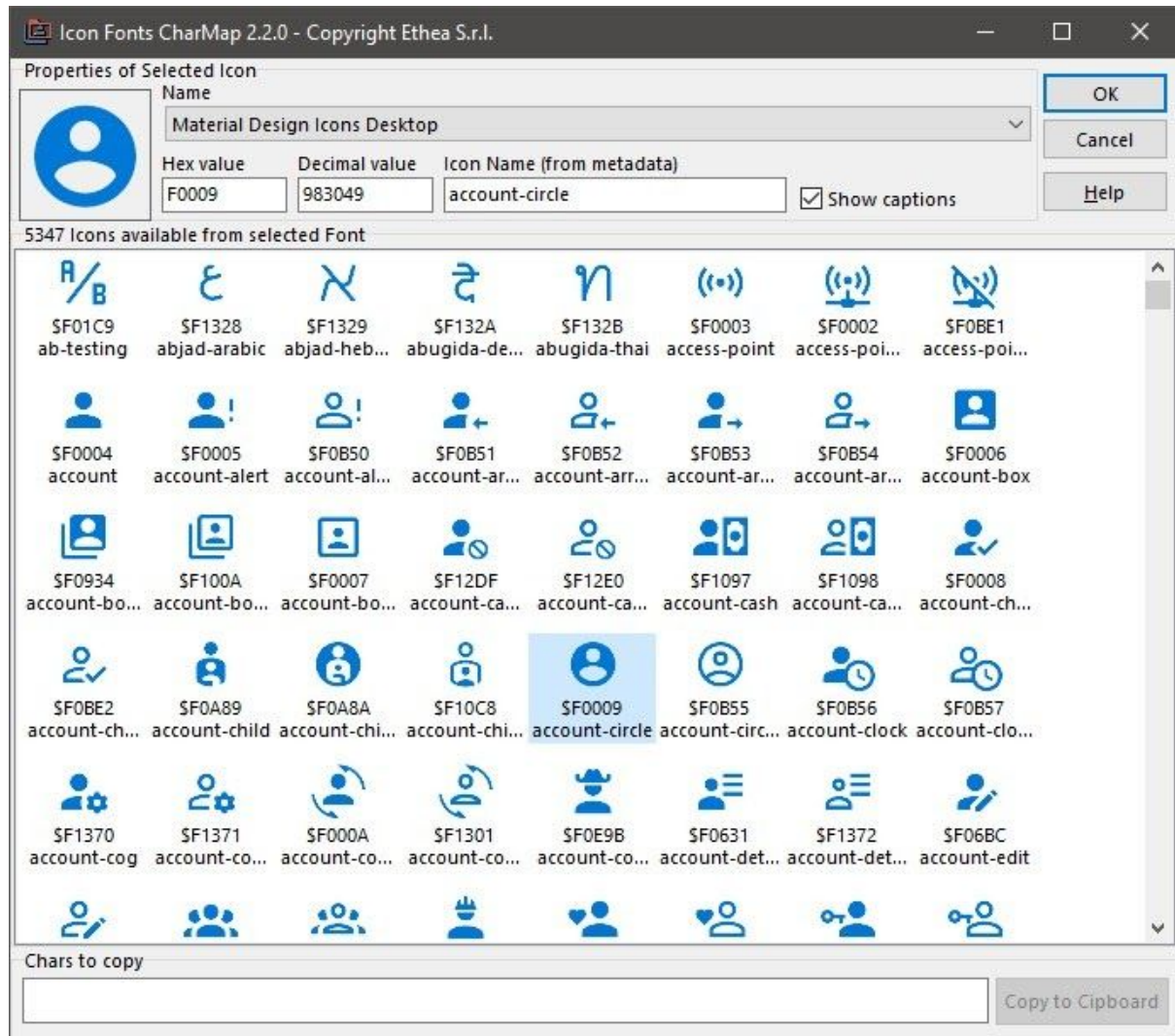


Utilities

Both libraries contain some utilities to ease configuration and setup of image collection.

IconFonts CharMap and Icons catalog

To ease setup of the Icons to collect, a “custom” CharMap clone application has been implemented. The system-provided one is not able to show some special “surrogate-pair” characters.

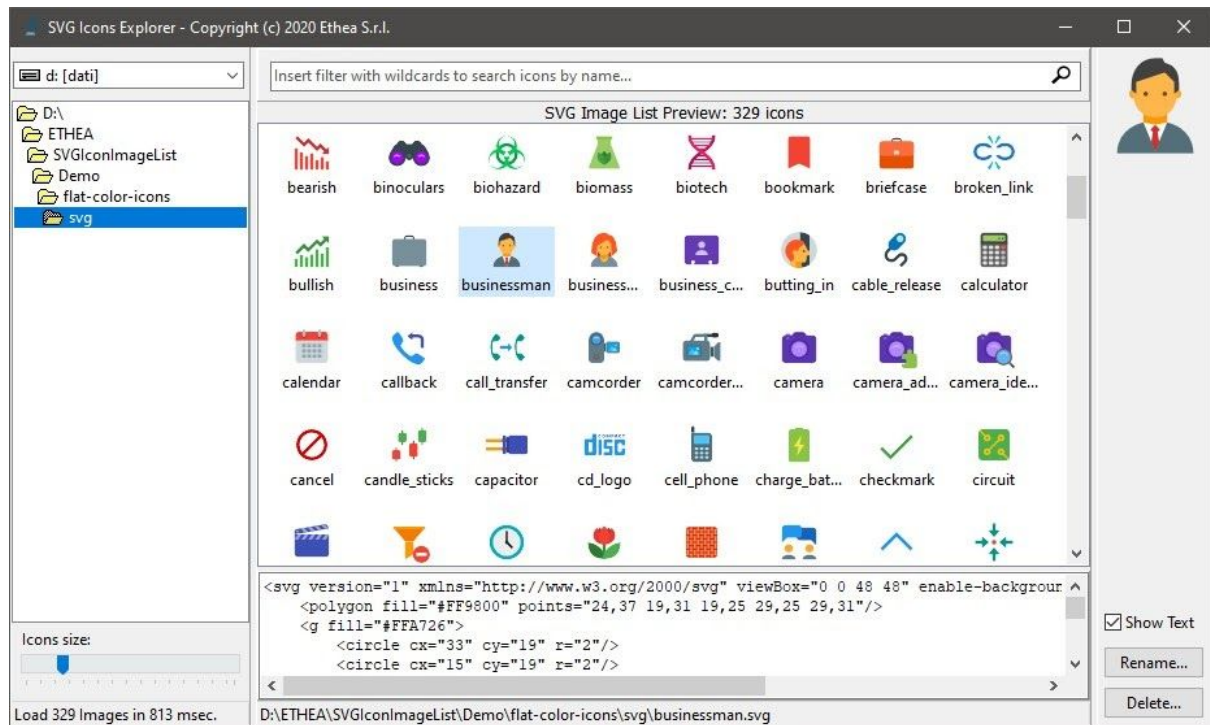


Within the CharMap application it is possible to select a different Font to be displayed, select icons to be imported into the collection, by pressing Ctrl + Click (and then OK).

Finally, there is also a chance to define a catalog of "metadata" about icons, to associate them with a name: in the above figure, the "Material Design Icons Desktop.ttf" font has been mapped into the Icons.MaterialDesign.pas unit unit generated by Andrea Magni's IconFontsImageListUtility (<https://github.com/andrea-magni/IconFontsImageListUtility>).

SVG Icons Explorer

To ease management of SVG icons, this utility has been implemented. It allows the developer to browse file-system and preview SVG files directly as Icons (handy as Windows Explorer currently has no support for this).



It is also possible to filter the icons, see the text version of the file and resize them.

Icons Explorer is an example of Delphi application using SVGIconImageList component, being a "single-window" application it is not necessary to follow the collection + virtual approach (discussed earlier in this document).

Demo projects and documentation

Both libraries are provided with an exhaustive Demo project showcasing main features:

- use of icons on TToolBar, TButton, TActionList / TAction, TTreeView and TListView;
- icons rescaling (through a trackbar);
- adapting the color to current VCL style;
- use of TIconFontImage or TSVGIconImage components to render a single icon (retaining scalability);
- effect of applying attributes to all icons (Color, GrayScale ...)

Beware: when the demo project is compiled with Delphi 10.3 or higher, it uses a native VirtualImageList instead of the one provided.

To delve into details, both libraries have a comprehensive "Wiki" section, available at the URLs:

<https://github.com/EtheaDev/IconFontsImageList/wiki>

<https://github.com/EtheaDev/SVGIconImageList/wiki>

Thanks

A special thanks goes to **Vincent Parrett** (FinalBuilder author) and **Kiriakos Vlahos** (PyScripter author) who effectively contributed to SVGIconImageList library. Success of these libraries is to be shared with them also, as they helped me during the transition from "normal" to "Virtual" and contributed to testing and bug-fixing.

Personally, I also want to thank Embarcadero: their Delphi product enabled me to make my passion for programming a profession. The company I founded (Ethea Srl - <https://ethea.it/>) is Technology Partner and also takes care of other Open-Source projects such as InstantObjects and Kitto (<https://github.com/EtheaDev>).

Firemonkey version

Yes, of both libraries there is also a version for FireMonkey... but we will see it in a future article.

Copyright © 2020 - Ethea Srl - all rights reserved