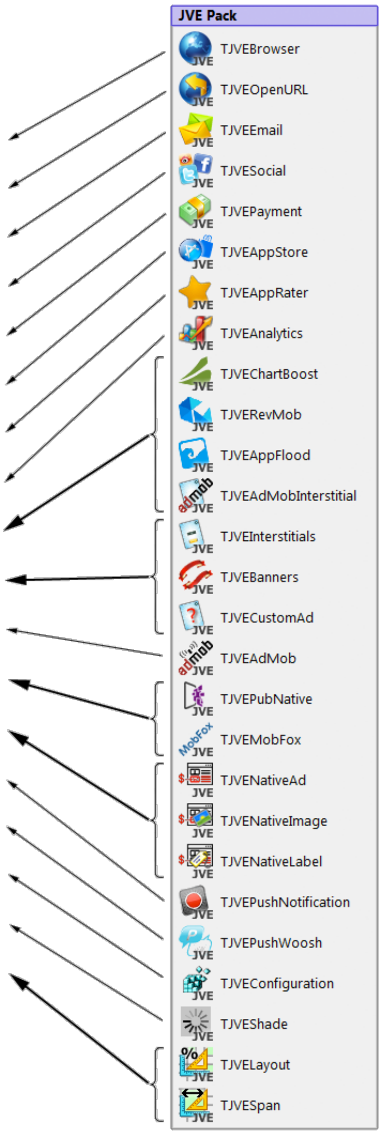
JVEsuite *by JVEsoft*

Cross Platform Infrastructure Component Suite
For Embarcadero Delphi XE4 – 11 Alexandria

Technical Manual



For any inquiries, write to **components@jvesoft.com**
or visit **http://www.jvesoft.com/wp/**

1. Component Suite Overview

This Delphi XE4 – 11 Alexandria cross platform component suite, by JVEsoft ltd., features 28 components (8 controls and 20 non-visual components) and provides access to a wide array of features, expected from modern applications.

The components provide the following general functions:

- Variable screen-size oriented layout
- Embedded browser
- Email sending
- Twitter, Facebook and SinaWeibo sharing
- In-App purchases (via AppStore, Google Play, Windows Store and/or PayPal)
- Viewing products in AppStore and Google Play
- App Rating reminders
- Google Analytics (available for both, FMX and VCL)
- Chartboost, AdMob and Facebook (Audience Network) interstitial advertising
- AdMob and Facebook (Audience Network) banner advertising
- MobFox and PubNative native advertising
- Generic banners, native and interstitials support
- Push notifications (raw requests or via PushWoosh service)
- Persistent storage and in-progress app shading

All the components can be used on all the platforms, supported by Delphi XE4 – 11 Alexandria (Android not being available in Delphi XE4; some functionality not available in XE5): iOS Device, Android, Mac OS X and Windows (Windows Store integration is only available since 10.3 Rio), except for Linux.

Not all features are supported on all platforms (for example advertising is only available on mobile), but fallback features are provided (fallback to in-house ads, for example) and no component will fail on a wrong platform.

All these components were thoroughly checked in Delphi XE4 – 11 Alexandria and running on a real Windows 7, 10 and 11, Mac OS X 10.8.4 – 11.6, various Android phones and tablets, iPad, iPhone and iPod Touch (however no warranty implicit or explicit is provided with this suite).

Here is what you will find inside the package (see *Package Content* section for details):

- Runtime package, compiled for Debug and Release for all platforms
- Design Time package, compiled for Debug and Release
- Well commented source code for the entire pack (about 20,000 lines)
- Where needed, instructions on making it work
- Original Chartboost, Audience Network and AdMob libraries
- Demo App – a simple app showing some of the modules of this pack in action

Though this document attempts to describe the functionality of the package, the source code of the components in the pack provides further input on their use (it is extensively documented).

2. *Starting Your Work*

Both iOS and Android platforms need further configuration of the project to make them work correctly. There are also special considerations when using In-App Purchasing mechanism on the OS X platform. These are listed below.

2.1 iOS

To use this library on iOS, there are some frameworks, which Delphi should be aware of, but by default isn't, so you need to perform the following changes in the IDE (you need to do this just once, it will apply to all projects):

1. Go to Tools → Options → SDK Manager → iOS Device, for each iPhoneOS device:

2. Click on any framework (just a click: there is a bug in IDE).

3. Click "Add a new path item" image button (top one)

a. Set "Path on remote machine" to "/System/Library/Frameworks" or "$(SDKROOT)/System/Library/Frameworks" (same as other frameworks).

b. Set "Framework name" to "`SystemConfiguration`".

c. Click Ok (nothing should be selected in the radio and leave the checkbox unchecked).

4. Repeat step 3 with Framework name "`AdSupport`", "`StoreKit`", "`Social`", "`CoreData`", "`CoreTelephony`", "`CoreMedia`", "`SystemConfiguration`", "`Security`", "`EventKit`", "`EventKitUI`", "`AVFoundation`", "`iAd`", "`AudioToolbox`", "`WebKit`", "`CoreBluetooth`", "`SafariServices`", "`CoreMotion`", "`MediaPlayer`", "`MobileCoreServices`", "`CoreVideo`", "`FileProvider`", "`CFNetwork`", "`MessageUI`", "`Metal`", "`ImageIO`", "`CoreAudio`", "`MediaToolbox`", "`CoreMIDI`", "`JavaScriptCore`", "`ModelIO`", "`IOSurface`", "`UserNotifications`", "`AVKit`", "`UniformTypeIdentifiers`", "`AppTrackingTransparency`" (no need to duplicate already existing items: Delphi XE5, for example, already includes `StoreKit` and `iAd`) and "AVFCore" (this framework should be defined with the "Path on remote machine" set to "$(SDKROOT)/System/Library/PrivateFrameworks").

5. Similarly add a new Library: choose "$(SDKROOT)/usr/lib" in Path on remove machine, "libcharset.1.tbd" in File mask and Library path in Path type.

6. Click "Update Local File Cache" button.

In each project, which is using this library, you need to perform the following changes:

1. Go to Project → Options → Delphi Compiler → Linking.

2. Choose Target "All Configurations – iOS Device platform".

3. Enter the following value in "Options passed to the LD linked" (this should be written as a single line; in some cases `-ObjC` linker option is also needed):

```
-ObjC -lz -framework SystemConfiguration -weak_framework Social
-framework StoreKit -framework MessageUI
-framework AudioToolbox -weak_framework AdSupport
-framework Security -framework CoreData -framework CoreTelephony
-framework AVFoundation -framework EventKit
-framework EventKitUI -framework CoreMedia
```

```
-framework CoreBluetooth -weak_framework SafariServices
-weak_framework CoreMotion -framework MediaPlayer
-framework CoreVideo -weak_framework MobileCoreServices
-weak_framework FileProvider -weak_framework WebKit
-weak_framework CFNetwork -weak_framework Metal
-weak_framework ImageIO -weak_framework CoreAudio
-weak_framework MediaToolbox -weak_framework CoreMIDI
-weak_framework JavaScriptCore -weak_framework ModelIO
-weak_framework AppTrackingTransparency
```

In addition to this the Receive Push Notifications should be selected within Project Option's entitlements list, if you want to use push notifications. If you want to use AdMob, add a new version info key for iOS platforms, called "`GADApplicationIdentifier`", with the value set to the app id supplied by AdMob.

- **Facebook Audience Network**

Due to the way Audience Network is compiled by Facebook, there is a cumbersome way of configuring it for iOS use.

1. FBAudienceNetwork.framework folder is included with this suite. Copy it to your Mac machine.

2. Within your Mac sign this framework with the exact same identity that you use for the rest of your application, by executing:

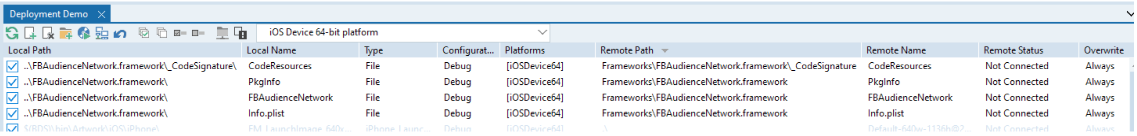
```
/usr/bin/codesign --deep -s identity -f framework
```

For example:

```
/usr/bin/codesign --deep
-s "iPhone Developer: Yegor Kurbachev (ABCD12E3FG)"
-f "/Users/Yegor/Desktop/Frameworks/FBAudienceNetwork.framework"
```

3. Copy the (updated, signed) files back to your PC.

4. Configure their deployment into the `Frameworks` folder, copying sub-folders hierarchy (your deployment record should look something like in the picture).



	Local Path	Local Name	Type	Configurat...	Platforms	Remote Path	Remote Name	Remote Status	Overwrite
☑	..\FBAudienceNetwork.framework_CodeSignature\	CodeResources	File	Debug	[iOSDevice64]	Frameworks\FBAudienceNetwork.framework_CodeSignature	CodeResources	Not Connected	Always
☑	..\FBAudienceNetwork.framework\	PkgInfo	File	Debug	[iOSDevice64]	Frameworks\FBAudienceNetwork.framework	PkgInfo	Not Connected	Always
☑	..\FBAudienceNetwork.framework\	FBAudienceNetwork	File	Debug	[iOSDevice64]	Frameworks\FBAudienceNetwork.framework	FBAudienceNetwork	Not Connected	Always
☑	..\FBAudienceNetwork.framework\	Info.plist	File	Debug	[iOSDevice64]	Frameworks\FBAudienceNetwork.framework	Info.plist	Not Connected	Always
☑	$(BDS)\bin\Artwork\iOS\iPhone\	FM_LaunchImage_640x...	iPhone_Launc...	Debug	[iOSDevice64]	.\	Default-640w-1136h@2...	Not Connected	Always

Notice, if you are working on several apps, which use different signing, you would need to repeat this process for each signing certificate, taking care deploying the correct set. In case of an invalid signature, the execution will fail with an error "Unable to install package (e800801c)".

For troubleshooting you may also check the iOS console log; if it presents FAN Error, then probably you have your signing procedure correct, but deployment wrong.

If you do not know the name of the correct identify, switch `PAServer` into verbose mode ("v" command) and execute the app. You will see codesign execution with the identity name.

- **iOS 14 Support**

To use advertising in iOS 14 you would need to add the following:

SKAdNetworkItems to your info.plist.TemplateiOS.xml file. See Demo app for an integration example.

Add version info NSUserTrackingUsageDescription key with the appropriate description (for example “This identifier will be used to deliver personalized ads to you.”).

You should call: JVERequestTrackingAuthorization function, if you intend to use AdMob; JVEFacebookSetAdvertiserTracking function, if you intend to use Facebook Audience Network (no ads will be presented if this function is never called); JVEChartBoostSetGDPRConsent, if you intend to use ChartBoost (for GDPR compliance).

2.2 Android

On Android further configuration of Android Manifest XML file is needed for specific components. When first compiling for an Android platform Delphi will create a new AndroidManifest.template.xml file. This is the file you should edit; the code should be added into the application tag.

- **AdMob**

Select AdMob Service within Project Option’s entitlements list, add jve-admob.jar file to the Libraries section of the Project Manager (under Target Platforms then Android) and add a new meta-data to your manifest file (with the actual value, as provided to you by AdMob):

```
<meta-data android:name="com.google.android.gms.ads.APPLICATION_ID"
android:value="ca-app-pub-9999999999999999~9999999999"/>
```

- **Chart Boost**

Add chartboost.jar and jve-chartboost.jar file to the Libraries section of the Project Manager (under Target Platforms then Android). The following should be added to the manifest file:

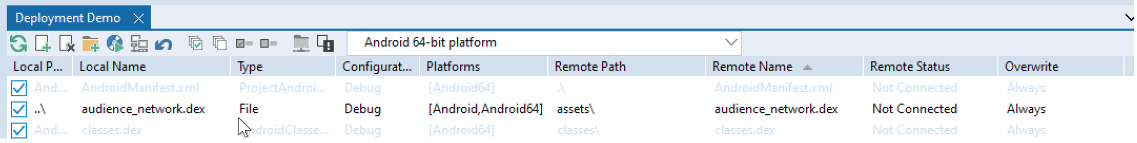
```
<activity android:name="com.chartboost.sdk.CBImpressionActivity"
android:excludeFromRecents="true"
android:hardwareAccelerated="true"
android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen"
android:configChanges="keyboardHidden|orientation|screenSize" />
```

- **Facebook’s Audience Network**

To use Audience Network on Android you need to perform the following three steps:

1. Add audience-network.jar file to the Libraries section of the Project Manager (under Target Platforms then Android).

2. Add `audience_network.dex` file to Project → Deployment, to be deployed within the `assets` folder (your deployment record should look something like in the picture).



	Local P...	Local Name	Type	Configurat...	Platforms	Remote Path	Remote Name	Remote Status	Overwrite
☑	And...	AndroidManifest.xml	ProjectAndroi...	Debug	[Android64]	.\	AndroidManifest.xml	Not Connected	Always
☑	..\	audience_network.dex	File	Debug	[Android,Android64]	assets\	audience_network.dex	Not Connected	Always
☑	And...	classes.dex	AndroidClasse...	Debug	[Android64]	classes\	classes.dex	Not Connected	Always

3. The following should be added to the manifest file:

```
<activity android:name="com.facebook.ads.AudienceNetworkActivity"
  android:configChanges="keyboardHidden|orientation|screenSize"
  android:excludeFromRecents="true" android:hardwareAccelerated="true"
android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen"
/>
```

- **Push Notifications**

Select Push Notifications within Project Option’s entitlements list and enter all the data within the Services tab.

- **Permissions**

Various components might require the following permissions to be selected within the Uses Permissions list of the Project Options:

- Access network state
- Access Wi-Fi state
- Get accounts
- Internet
- Read external storage
- Read phone state
- Read user dictionary
- Vending billing
- Wake lock
- Write external storage

2.3 OS X

Requirements for Mac OS in-app purchases make it incompatible with debugging from within Delphi: the first app execution must be from Finder and since with Delphi each execution is a first execution, Delphi will not be able to debug the app.

If you are using in-app purchases, you will not be able to run the RELEASE build from within Delphi and you will not be able to submit DEBUG build to Mac App Store. In other words, you cannot debug in-app purchases in Delphi. You can run the application locally on your Mac and, if built in RELEASE configuration, you will be able to perform in-app purchases, but not debug the process.

For Mac OS 64-bit support, you need to import several Mac OS frameworks. Follow the same steps, given for iOS above, but for the following frameworks: "SystemConfiguration", "StoreKit" and "WebKit".

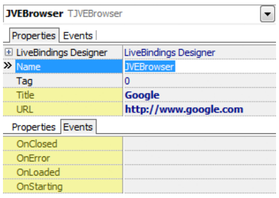
Also, similarly to iOS, you would need to provide Mac OS 64-bit "Options passed to the LD linked", as follows:

```
-framework SystemConfiguration -framework StoreKit -framework WebKit
```

3. *Embedded Browser*

File: `JVE.Browser.pas` Inheritance: `TComponent` ← `TJVEBrowser`

This component allows you to open a web browser in a popup modal dialog. The list of properties exposed is:



JVEBrowser TJVEBrowser

Properties | Events

LiveBindings Designer	LiveBindings Designer
Name	JVEBrowser
Tag	0
Title	Google
URL	http://www.google.com

Properties | Events

OnClosed	
OnError	
OnLoaded	
OnStarting	

- `URL` – this is the URL to open; you must provide a scheme for this component to always work correctly (for example "http://").
- `Title` – the title to give the browser dialog.

The events exposed are:

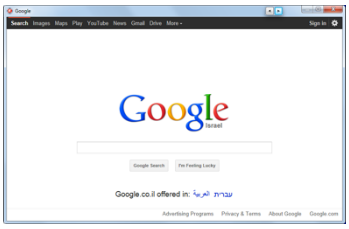
- `OnStarting` – occurs before a page begins to load (for example when clicking a link). You can use it to check the URL and decide whether the click should be allowed or ignored or even if the browser should be closed.
- `OnLoaded` – occurs when the page was successfully loaded.
- `OnError` – occurs on loading errors. Default behavior is to present an error and close the browser.
- `OnClosed` – occurs when the browser was closed (by the user or via `OnStarting`).

The methods available in this component are: `Open`, `Close` and `IsOpen` (pretty self explanatory) and `OpenURL(URL,Title)` – convenience class procedure, which constructs `TJVEBrowser`, presents it and frees it, after the user closed it.

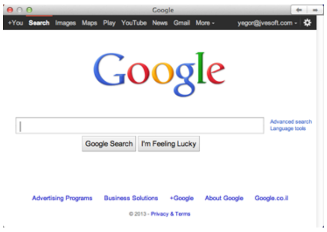
The component suite is also supplied with a legacy Delphi versions compatible variation of this component, using VCL.

The presentation of the browser dialog is platform dependent:

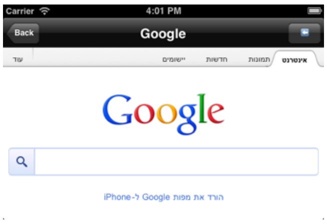
MS Windows



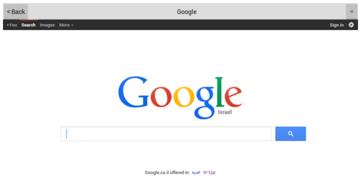
Mac OS X



iOS Device and Simulator



Android



4. Opening URLs

File: `JVE.OpenURL.pas` Inheritance: `TComponent` ← `TJVEOpenURL`

This component opens the default app, associated with the scheme on the platform (i.e. “http://” will open a browser; “ftp://” will open an FTP client and a custom scheme will be opened by the app, defined as supporting it.

This component defines a single property `URL` (an address to open), no events and the following instance methods:

- `Open` – opens the URL; returns True, if successfully opened.
- `CanOpen` – checks whether the URL can be opened (i.e. scheme is supported).

In addition to this, it provides the following class methods:

- `OpenURL(URL)` and `CanOpenURL(URL)` – convenience: as above, but without creating a component.
- `SchemeSupported(Scheme)` – checks whether the scheme is (i.e. “http”, “ftp”, etc.) is supported by the system and can be opened.
- `URLEncode(Value)` – convenience function: encodes the string to make it safe for URL inclusion (i.e. “Set to 40%” will become “Set%20to%2040%25”).

When working with this class, you should always use the full URL, including the scheme (otherwise it might fail on some platforms and the defaults used might not be consistent).

Special class methods are also provided in this component, providing an easy way to easily download the content of the provided URL (these function should only be used with HTTP and HTTPS protocols).

These functions download the content directly into a memory stream or a bitmap, making them easy to use, but they should not be used on large files. The function blocks the calling thread until they finish, so calling it on a secondary thread is recommended.

The functions are defined as follows (several overloads are given to achieve this):

- `DownloadURL(URL: String[, Headers: TStrings][, var Error: String]): TMemoryStream;`
- `DownloadBitmap(URL:String[,Headers: TStrings][,var Error: String]): TBitmap;`
- `TBitmap.Download(URL:String[,Headers:TStrings][,var Error:String]): TBitmap;`

The function returns the filled memory stream, bitmap or **nil**, if an error occurred. An optional error parameter provides a user readable message; optionally extra HTTP headers can be provided as well.

If planning to use these functions on multiple platforms, it is important to free the resulting class after assigning the value. Notice, due to a bug in Delphi, `DownloadBitmap` synchronizes with the main thread to decode the image (if the main thread is stuck, this call will also stuck).

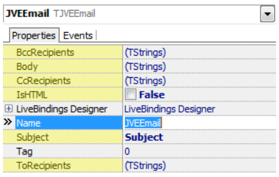
These functions use appropriate OS functions, instead of trying to support download itself or using the Indy library. As such they also allows for SSL connectivity without using additional encryption libraries on all platforms.

5. *Sending Emails (“Tell a Friend”)*

File: `JVE.Email.pas` Inheritance: `TComponent` ← `TJVEEmail`

This component provides a way for the user to send an email with some default content. Native interface is used, if available, otherwise the default email application is used. The properties exposed by this component are:

- `Subject` and `Body` – default subject and body for the email.
- `IsHTML` – specifies whether the `Body` is in plain text format or HTML.
- `ToRecipients`, `CcRecipients` and `BccRecipients` – default list of recipients for the email.



JVEEmail TJVEEmail

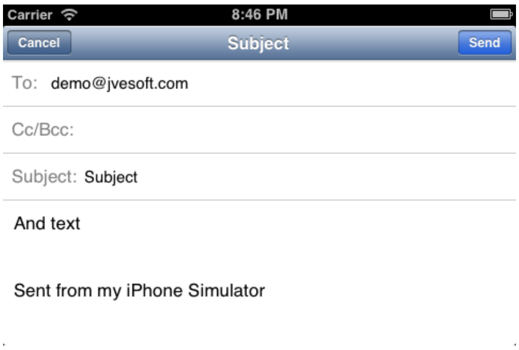
BccRecipients	(TStrings)
Body	(TStrings)
CcRecipients	(TStrings)
IsHTML	□ False
⊞ LiveBindings Designer	LiveBindings Designer
» Name	JVEEmail
Subject	Subject
Tag	0
ToRecipients	(TStrings)

While no events are exposed, the following instance methods are available:

- `Send` – opens the email composition window, allowing the user to send the email.
- `CanSend` – use to check whether native interface for email composition, if any, is available. Even if this function returns `False`, you can still call `Send`, default email application will then be used as fallback. On platforms, which do not have a native interface, this function always returns `True`.

The native interface is available in iOS and Android; for other platforms default email application is used. Example presentation is as follows:

iOS Device and Simulator



To: demo@jvesoft.com

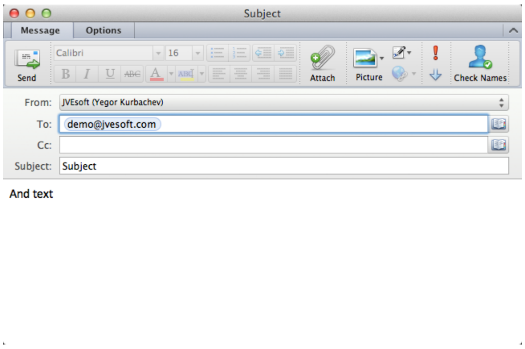
Cc/Bcc:

Subject: Subject

And text

Sent from my iPhone Simulator

Other Platforms (for example)



From: JVEsoft (Yegor Kurbachev)

To: demo@jvesoft.com

Cc:

Subject: Subject

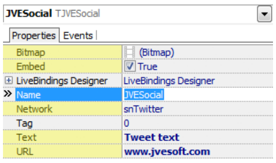
And text

6. *Social Networks Support*

File: `JVE.Social.pas` Inheritance: `TComponent` ← `TJVESocial`

This component allows the user to tweet or post to one of the supported networks; similarly to TJVEEmail, this component opens a dialog, through which the user can enter his message, while the default is provided via component properties. The properties exposed are:

- `Network` – chooses social network to post to: `snTwitter`, `snFacebook` and `snSinaWeibo` are supported.
- `Text` – the default text for the post.
- `URL` – any hyperlink to include with the post.



JVESocial TJVESocial

Bitmap	(Bitmap)
Embed	☑ True
⊞ LiveBindings Designer	LiveBindings Designer
» Name	JVESocial
Network	snTwitter
Tag	0
Text	Tweet text
URL	www.jvesoft.com

- `Bitmap` – an image to include with the post. This option is only supported in iOS.
- `Embed` – if web fallback is used, specifies whether the browser should be opened embedded in the app (i.e. `TJVEBrowser`) or externally (i.e. `TJVEOpenURL`).

While no events are exposed, the following methods are available:

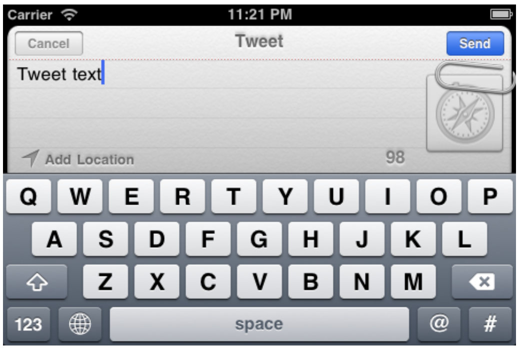
- `Open` – opens the posting dialog (or external web browser, if requested).
- `CanOpen` – use to check whether native posting interface, if any, is available. Even if this function returns `False`, you can still call `Open`, web browser fallback will then be used. On platforms, which do not have a native interface, this function always returns `True`.
- `CanOpen(Network)` – this class function works the same way as `CanOpen`, but does not need an instance to be created.

The native interface is only currently available in mobile platforms; for other platforms an appropriate web page will be opened.

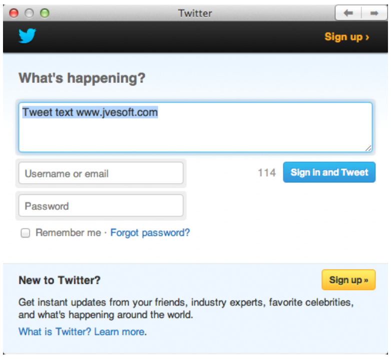
It should be noted that for circumstances beyond the author’s control, this component is much more stable (with regard to native interface) on real iOS devices, sometimes failing within the iOS simulator.

Example presentation (given for Twitter, other networks looks similarly) is as follows:

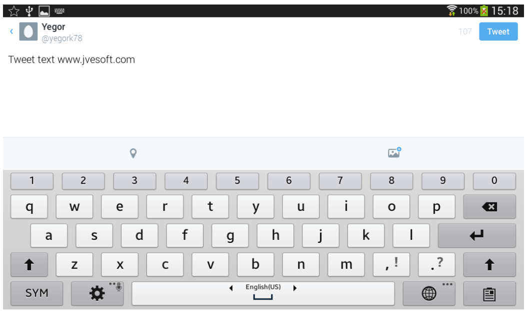
iOS Device and Simulator



Other Platforms (for example)


Twitter
Sign up ›
What's happening?
Tweet text www.jvesoft.com
Username or email
114
Sign in and Tweet
Password
Remember me · Forgot password?
New to Twitter?
Sign up »
Get instant updates from your friends, industry experts, favorite celebrities, and what's happening around the world.
What is Twitter? Learn more.

Android



7. Payments Library

File: `JVE.Payment.pas` Inheritance: `TComponent` ← `TJVEPayment`

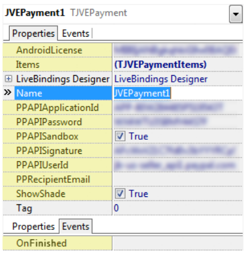
This component allows the user to request payments from the user. It supports native payments, made through the AppStore (on iOS and Mac OS X) and Google Play (on Android; not available in Delphi XE5), and PayPal payments (on all platforms).

Unlike other classes, due to Apple's requirements (i.e. digital goods **must** be bought via an in-app payments; non-digital goods must **not** be an in-app purchase), this class will **not** fallback to PayPal automatically. To use PayPal on a mobile platform, the payment item must define PayPal in its Platforms list and it must **not** define the mobile platform.

The component is capable of managing all In-App Payments required by an application and it is usually more efficient to have one such component taking care of all the payments, rather than creating separate components for various payments. Notice, this component works with remote servers, so processing is mostly **not** immediate (sometimes taking several seconds to complete).

You can still create multiple instances of this component, if, for example, you need to create PayPal payments to varying recipients.

The diagram of properties, this component exposed is as follows:



JVEPayment1 TJVEPayment

Properties | Events

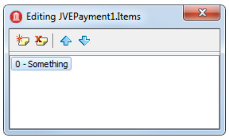
AndroidLicense	[redacted]
Items	(TJVEPaymentItems)
⊞ LiveBindings Designer	LiveBindings Designer
» Name	JVEPayment1
PPAPIApplicationId	[redacted]
PPAPIPassword	[redacted]
PPAPISandbox	☑ True
PPAPISignature	[redacted]
PPAPIUserId	[redacted]
PPRecipientEmail	
ShowShade	☑ True
Tag	0

Properties | Events

OnFinished	

Items is a collection

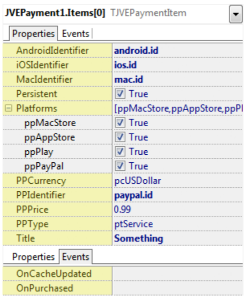
←



Editing JVEPayment1.Items

0 - Something

←



JVEPayment1.Items[0] TJVEPaymentItem

Properties | Events

AndroidIdentifier	android.id
iOSIdentifier	ios.id
MacIdentifier	mac.id
Persistent	☑ True
⊟ Platforms	[ppMacStore,ppAppStore,ppPl
ppMacStore	☑ True
ppAppStore	☑ True
ppPlay	☑ True
ppPayPal	☑ True
PPCurrency	pcUSDollar
PPIdentifier	paypal.id
PPPrice	0.99
PPType	ptService
Title	Something

Properties | Events

OnCacheUpdated	
OnPurchased	

The properties of each payment are as follows:

- `iOSIdentifier` – the unique identifier of the payment, to be used to access Apple Store from an iOS device.
- `MacIdentifier` – the unique identifier of the payment, to be used to access Apple Store from a Mac device.
- `PPIdentifier` – the unique identifier of the payment, used for PayPal: could be any unique identifier of the payment, for the item (unique for the user, not globally unique across users).
- `WindowsToken` – the unique identifier of the payment, to be used to access Windows Store.
- `Persistent` – specifies whether the item is persistent (buy once, use forever or "non-consumable"), like "unlock all levels", or consumable (can be bought several times), like "3 more lives". Android products are always persistent; use `Consume` procedure to consume them.
- `Platforms` – the list of platforms, which support this payment.
- `Title` – this is the PayPal item; for additional use, see below section *Caching Prices*.
- `PriceCache` – a read/only property; see below section *Caching Prices*.

- PPCurrency and PPPrice – the price and currency of the payment, for PayPal use. Since PayPal does not allow products to be predefined, you need to supply the amount to charge for each payment. You should keep this figure accurate, because it is also used in *Google Analytics* reporting, when the real price of the item is not known.
- PPType – specifies the type of the item: ptGoods, ptServices or ptDigitalGoods. Notice, as of the writing of this document, PayPal used a special site for Digital Goods, which failed miserably in most browsers, so its use is strictly discouraged.
- Purchased – a read/only property, which indicates whether a Persistent item was purchased. Notice, this property does not contact the server to verify the purchase, only local storage; see RestorePurchases method for that function.

For the component itself, the properties are:

- Items – the collection of In-App Payments this component exposes.
- PPAPIApplicationId, PPAPIUserId, PPAPIPassword, PPAPISignature and PPAPISandbox – these properties control the way the control connects to PayPal services; you don't need to change them, if you are not using PayPal payments. See *Configuring PayPal Payments* section below for more details.
- PPRecipientEmail – the email of the payment recipient.
- AndroidLicense – the license key of the app, as generated within the available console.
- ShowShade – specifies whether the app is shaded, when it processes requests. See *In-Progress App Shading* section below.

Only one method is provided for the item: Buy, which initiates the purchase process for the item. The process runs in background; any inquiries needed are presented in modal dialogs. See events section below to learn how to respond to issues in this process.

The following methods are provided for the component:

- CanBuyNatively – this class function returns whether buying natively is supported. On platforms, which only support PayPal always returns True.
- Buy(Id) – a convenience method: calls the Buy method of the item, identified by is Identifier; returns True if the identifier was found, returns False otherwise.
- RestorePurchases – restores any previously made purchases (only for non-consumable items), see *Persistent Products* for more details.
- CachePrices – See *Caching Prices* section below.
- IsBusy – Indicates whether the component is currently busy restoring or buying some items; OnFinished will be called when done.

The following events are exposed by this component and by items:

- OnFinished – called after a buy or restore purchases request is finished processing. This method is guaranteed to be called after every Busy period. The default behavior is to present an error message, unless the Status returned is psSuccess or psCancelled.
- OnPurchased – called when an item was purchased (or its purchase restored). You must always listen to this event, even if you have not requested a purchase (if previous session did, but failed, Apple might deliver the product in the next session without any request).
- OnPriceCached – see Caching Prices.

7.1 Persistent Products

The fact that a persistent product was purchased is stored in the local persistent storage, so you can use the Purchased property of the item, which immediately returns, whether the item is purchased (you don't need to manage this yourself).

This persistent storage is based on the Identifier of the item, which should not be changed in the lifetime of the application for this feature to work correctly.

Notice, this persistent storage is by no means cryptographically strong. It is generally strong enough against an advanced user (i.e. copying registry from one machine to another, for example, will not help), but it does not attempt to be strong enough to withstand a hacker attack (code disassembly and/or debug will reveal a way for the particular app to be broken).

If you need higher level of protection or if you need further validations against the payment servers, you might need to use custom cryptographic libraries or other tools.

Once the app is uninstalled and reinstalled, the persistent storage will no longer contain the fact that the purchase was made. Generally speaking, another purchase attempt (if the item is defined as `Persistent`) will not result in an additional charge, instead the item will be reported as bought, without any further debit.

You can (and probably should) provide the user with an option to automatically restore any previous purchases (using the `RestorePurchases` method). This method queries the server for the items, which were already purchased.

For Apple In-App Purchases, non-consumable items are restored if Apple user already purchased the item; for PayPal, the items are restored, if they were already bought using this hardware (this feature uses all of the computer's MAC Addresses to perform the best possible match, even if some of the hardware was changed).

It should be noted that PayPal does not generally provide this feature and PayPal's other capabilities were used to provide it.

7.2 Caching Prices

This component is capable of retrieving prices and localized titles of products from the server, for products defined on Apple platforms. It also formats prices in a user-friendly manner for PayPal items.

To start this process, call the `CachePrices` method. This process runs in background and returns immediately.

Once item details are updated, `PriceCache` and `Title` properties will be updated (`Title` will not be updated when using PayPal) with their localized values and `OnPriceCached` will be called (you can use it to update the user interface, if needed).

In case an item was reported as invalid, the `PriceCache` field will contain a single dash (i.e. "`-`"), instead of the actual price, but no error will be reported at this point (an error is only reported once you call the `Buy` method).

Notice, you should always be prepared for `OnPriceCached` event to be called: it might be called, for example, during the purchase processing. Calling `CachePrices` also speeds up any future calls to the `Buy` method.

7.3 Configuring Apple's In-App Payments

This document assumes basic knowledge of In-App Payments on Apple platforms. To use Apple's In-App Purchases:

- The app must be created in iTunesConnect
- It must **not** use a wildcard bundle identifier
- In-App Payments items should be created for the app in iTunesConnect
- Delphi project should reference the correct bundle id (in Project → Options → Version Info → iOS Device / Simulator platform → `CFBundleIdentifier`).
- Exact Product ID of each In-App item should be used within the payment item's Identifier property

7.4 Configuring PayPal Payments

On any platform you need valid PayPal Adaptive Payments credentials (the ones supplied with this component as default (and visible in the screenshots), are published sandbox credentials, which cannot be used in production).

Since PayPal stopped issuing Adaptive Payments credentials, you might not be able to use PayPal in your application. Currently PayPal integration should be considered legacy functionality.

Getting the `PPAPIApplicationId`:

- Go to `http://apps.paypal.com` and login.
- Click New App under My Applications and fill your app data (important: tick Basic Payments under Adaptive Payments). If you don't choose to many extra items in that screen, your app should be automatically approved.
- On the main screen you will now have your Sandbox ID and Live App ID, set the `PPAPIApplicationId` property to one of them (set `PPAPISendbox` appropriately).

Getting the `PPAPIUserId`, `PPAPIPassword` and `PPAPISignature`:

- Go to `http://www.paypal.com` and login.
- To use PayPal API you must have a Premier or Business PayPal account (you can upgrade Personal to Premier account for free in a minute by clicking Upgrade next to Account Type field on My Account → Overview; you can also have separate accounts for this).
- Go to My Account → Profile and choose API Access under Account Information.
- Choose Option 2 – Request API credentials.
- Choose the left option – Request API signature and click Agree and Submit.
- Copy the API Username, API Password and Signature to the matching properties, then click "Done".

7.5 Configuring Android's In-App Payments

This document assumes basic knowledge of In-App Billing in Google Play. To use payment the following should be done:

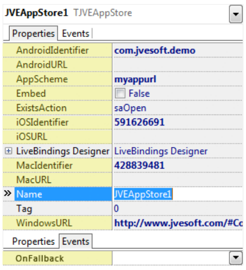
- The app must be created in Google Play.

- The app must be compiled with valid certificates.
- A beta binary must be uploaded with billing permissions.
- In-App products should be created.
- A license key property of the component should be set.
- Android testing should be done with a Google account, having permission to test.
- Exact Product ID of each In-App item should be used within the payment item’s Identifier property.

8. App Store Presentation

File: `JVE.AppStore.pas` Inheritance: `TComponent` ← Inner Class ← `TJVEAppStore`

This component allows the user to view an app in the app store (could be used for cross promotions, for example). In case the app is available for several platforms, the component can choose the best app to suggest. The following properties are exposed:



JVEAppStore1 TJVEAppStore

AndroidIdentifier	com.jvesoft.demo
AndroidURL	
AppScheme	myappurl
Embed	□ False
ExistsAction	saOpen
iOSIdentifier	591626691
iOSURL	
⊞ LiveBindings Designer	LiveBindings Designer
MacIdentifier	428839481
MacURL	
» Name	JVEAppStore1
Tag	0
WindowsURL	http://www.jvesoft.com/#Cc

OnFallback	

- `iOSIdentifier` and `MacIdentifier` – Apple ID of the app in the iOS / Mac Apple store.
- `AndroidIdentifier` – Play Market app identifier (for example `com.jvesoft.app`).
- `iOSURL`, `MacURL` and `AndroidURL` – URL of the app in the iOS / Mac Apple Store or Play Market. This will force the component to use web fallback, instead of native functionality.
- `WindowsURL` – URL of the Windows version of the app.
- `Embed` – if web fallback is used (or on Windows), indicates whether the URL should be opened embedded or in external browser.
- `AppScheme` and `ExistsAction` – see below for the treatment in relation to already installed apps.

This component also exposes an `OnFallback` event: when you ask this component to present an app, if it is not available on the current platform, the component will call this event to query for a fallback platform (return `spNone` to skip presentation entirely). If you don’t implement this event, the component will cycle through all the available platforms.

While no events are exposed, the following methods are available:

- `Open` – opens the app store with the product requested.
- `ActivePlatform` – this function returns the platform, for which the component will open the store, when `Open` function is called (this function might call `OnFallback` event).

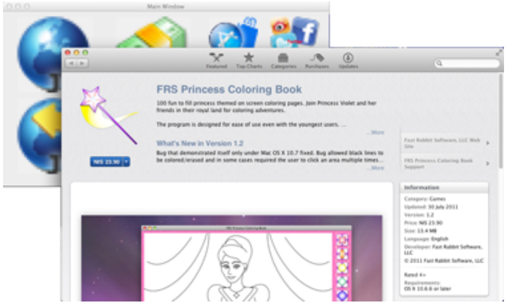
It should be noted that due to platform limitations, it is impossible to open a Mac App Store to view an app while running in iOS. Since App Store is not fully available in iOS Simulator (and because of other limitations – see source code), this component is not fully functional in the iOS Simulator, to view iOS apps (it will open a blank browser, but won’t fail beyond that).

Since you would not normally present app store for an app, already installed by the user, this component can check whether the app to be advertised is already installed. To use this feature the target app should register some (unique) URL scheme it supports.

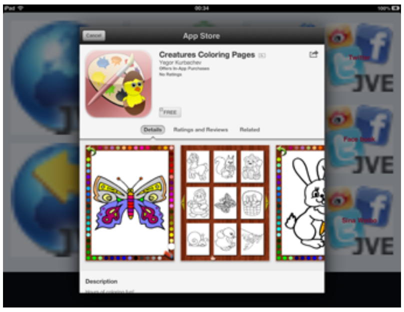
The component then checks whether there is an app supporting this scheme (you set the scheme to check using the `AppScheme` property). If an app is installed, the component uses the `ExistsAction` property to decide what to do: open app store (`saStore`), open the app itself (`saOpen`) or do nothing (`saIgnore`).

While showing an app from another platform, the component will always use a browser (embedded or external), but on Apple platforms it will present an app store, if it can. Example presentations are as follows:

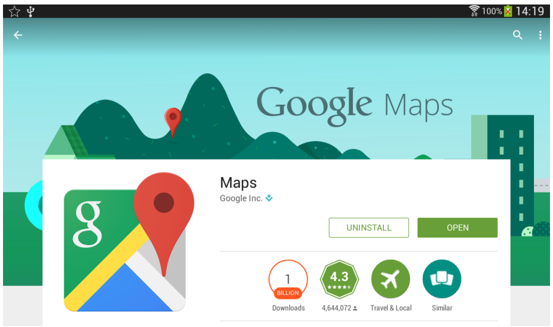
Mac OS X



iOS Device



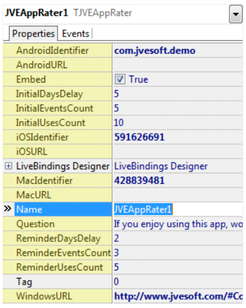
Android



9. *App Rating Reminder*

File: `JVE.AppStore.pas` Inheritance: `TComponent` ← Inner Class ← `TJVEAppRater`

This component tracks user behaviour and allows him to rate the app, once certain conditions are met (similarly to `Appirater` tool for iOS). The following properties are exposed:



JVEAppRater1 TJVEAppRater

Properties | Events

AndroidIdentifier	com.jvesoft.demo
AndroidURL	
Embed	☑ True
InitialDaysDelay	5
InitialEventsCount	5
InitialUsesCount	10
iOSIdentifier	591626691
iOSURL	
⊞ LiveBindings Designer	LiveBindings Designer
MacIdentifier	428839481
MacURL	
» Name	JVEAppRater1
Question	If you enjoy using this app, wo
ReminderDaysDelay	2
ReminderEventsCount	3
ReminderUsesCount	5
Tag	0
WindowsURL	http://www.jvesoft.com/#Cc

- `iOSIdentifier`, `iOSURL`, `MacIdentifier`, `MacURL`, `AndroidIdentifier`, `AndroidURL`, `WindowsURL` and `Embed` – all have the exact same function as in the `TJVEAppStore` component.
- `InitialDaysDelay` – number of days, to pass before the app queries the user for the first time. Should normally be at least 1.
- `InitialUsesCount` – number of times the user should run the app, before the first query.
- `InitialEventsCount` – number of major events, which should occur, before first query.
- `ReminderDaysDelay` – if the user decided not to rate the app, number of days, before he is reminded again. Should normally be at least 1.
- `ReminderUsesCount` – number of times the app runs, before the user is reminded.

- `ReminderEventsCount` – number of major events, before the user is reminded.
- `Question` – this is the question, which will be presented to the user (the question will not be asked in Native mode in iOS 10.3 and on, due to iOS implementation choices).
- `NativeiOS` – indicates the GUI version to use on iOS: 10.3, if available, 6.0 or external app store. Notice, by Apple's design, in 10.3 GUI there is no guarantee that the rating dialog will be presented at all, thus don't use this option as a response to a button click.

While no events are exposed by this component, the following methods are provided:

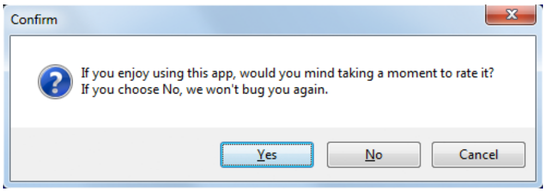
- `Open` – opens the app in app store for rating entry immediately (overriding usual delay and without a confirmation dialog). For the normal functionality, use `ProcessMajorEvent`.
- `QueryForFeedback` – if user has not requested not to be bugged, presents a dialog asking him, whether he would like to rate the app. If confirmed, you can call the `Open` function.
- `ResetDontAsk` – if the user chose not to ask him again, this would reset this selection. You should only use this in outstanding circumstances, for example, if you are out with a new app version and want the user to reconsider.
- `ProcessMajorEvent(OpenQuery)` – use this function to notify the App Rater that it is an appropriate time to show the Rate request. The popup will only be opened, if appropriate time has passed and other requires are met.

This last method is the **only** method you should normally call in this class. Pass `False` in `OpenQuery` to check applicability without opening a query. Returns `True`, if the interstitial query was indeed shown (or should now be shown, if requested not to open; counters are reset anyway in this case, so this component could be used as an arbitrary counter, if needed).

The normal use of this component is to configure it (in design time), then simply call the `ProcessMajorEvent` every time the user finishes a level or something similar. See also the `Interstitials` section for more use cases.

The query presented to the user is similar to the screenshot below, while the app presentation generally follows the way `TJVEAppStore` works (except that the Rating and Reviews tab is opened by default, if supported by the platform). Example presentations are:

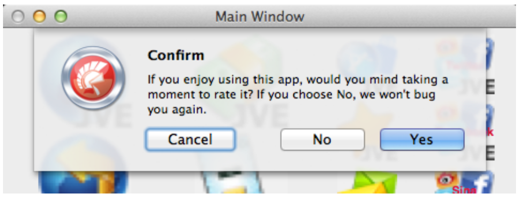
MS Windows



Confirm

If you enjoy using this app, would you mind taking a moment to rate it?
If you choose No, we won't bug you again.

Mac OS X

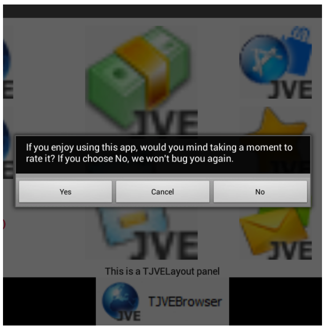


Main Window

Confirm

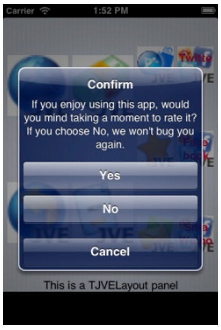
If you enjoy using this app, would you mind taking a moment to rate it? If you choose No, we won't bug you again.

Android



If you enjoy using this app, would you mind taking a moment to rate it? If you choose No, we won't bug you again.

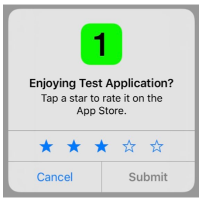
iOS Device and Simulator Query



Confirm

If you enjoy using this app, would you mind taking a moment to rate it? If you choose No, we won't bug you again.

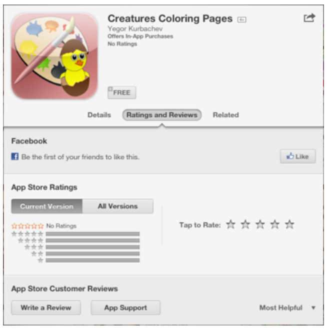
iOS 10.3 GUI Presentation



Enjoying Test Application?

Tap a star to rate it on the App Store.

iOS 6.0 GUI Presentation



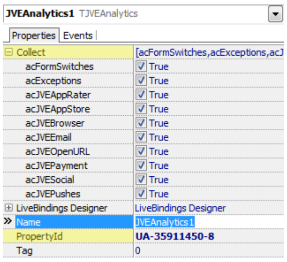
10. Google Analytics

For FMX apllications, use: File: `JVE.Analytics.pas` Inheritance: `TComponent` ← Inner Class ← `TJVEAnalytics`

For VCL applications, use: File: `JVE.Analytics.VCL.pas` Inheritance: `TComponent` ← Inner Class ← `TJVEAnalyticsForVCL`

This component links your application to the Google Analytics service. Use it if you want to track app usage statistics (you might need to get user's permissions before gathering his information; unless specifically coded, this component does **not** send user-identifiable information). The following properties are exposed by this component:

- `Active` – indicates whether analytics reporting is enabled.
- `PropertyId` – the property, defined within the Google Analytics service. See https://analytics.google.com/ for details. The property must be created as a Web, **not** an App (it will be capable of tracking all platforms nonetheless).
- `Collect` – specifies the type of events, which the component will keep track of, automatically. See below.



JVEAnalytics1 TJVEAnalytics

⊟ Collect	[acFormSwitches,acExceptions,acJ
acFormSwitches	☑ True
acExceptions	☑ True
acJVEAppRater	☑ True
acJVEAppStore	☑ True
acJVEBrowser	☑ True
acJVEEmail	☑ True
acJVEOpenURL	☑ True
acJVEPayment	☑ True
acJVESocial	☑ True
acJVEPushes	☑ True
⊞ LiveBindings Designer	LiveBindings Designer
» Name	JVEAnalytics1
PropertyId	**UA-35911450-8**
Tag	0

This component does not expose any events and the methods it is exposing are directed specifically to tracking data to servers.

See below for *Manual Tracking* and *Automatic Tracking* of events. The component is implemented to cache events and send them out 1 per second in a parallel thread. It also uses a persistent first-in-first-out queue. All in all this means that performance will not be affected and no further considerations need to be taken into account, when using this component.

Notice, Google Analytics tracking is **not** immediate! Several hours might be required for the data to be correctly aggregated and presented within the Google Analytics site.

This component is available for VCL and for FMX, though for VCL it is only compatible with Delphi XE8 and later.

10.1 Manual Tracking

The following methods are exposed by this component to help you send tracking events to the Google Analytics servers:

- `TrackScreen(Screen)` – use this to track a change in the current screen, i.e. another screen being opened.
- `TrackEvent(Category, Action, Label = '', Value = 0)` – use this to track arbitrary events; level and value figures are optional.
- `TrackSocial(Network, Action, Target)` – use this to track events, which could be considered to be social networks oriented.
- `TrackException(Exception, Fatal = True)` – use this to track exceptions or other errors. Set `Fatal` to `True` to indicate that the error was fatal.
- `TrackTransaction(TransactionId, Affiliation, Revenue, Shipping, Tax, Currency)` – this function can be used to start a payment transaction; you list the transaction content using the next function. The `TransactionId` should be unique.
- `TrackItem(TransactionId, ItemName, Price, Quantity, Code, Category, Currency)` – allows you to list the transaction content; the `TransactionId` of the items must be the same as passed to the `TrackTransaction` method.

In addition to the procedures listed above, there is the same set, prefixed with Broadcast (instead of Track), which performs the same action, but sends the information to all active TJVEAnalytics components, not just the current one (these are class procedures).

See Google Analytics website for more information about tracking your application. Helper routines are also available to transmit raw data to the servers (see component source code for more details).

10.2 Automatic Tracking

In addition to providing facilities to report events, the component is also capable of reporting various events on its own. To control this behaviour, the Collect property can be used, providing the following options:

- acFormSwitches – tracks when a new form becomes active.
- acExceptions – tracks all uncaught exceptions (see below for more details).
- acJVEAppRater – tracks activations of TJVEAppRater control.
- acJVEAppStore – tracks activations of TJVEAppStore control.
- acJVEBrowser – tracks all embedded browser executions (including the URL).
- acJVEEmail – tracks activations of TJVEEmail control.
- acJVEOpenURL – tracks all external browser executions (also tracks the URLs themselves).
- acJVEPayment – tracks all TJVEPayment payments as app revenues.
- acJVESocial – tracks activations of TJVESocial control.
- acJVEPushes – tracks all push notification registrations and pushes.

If you ever use the acExceptions option, the component will override the global Application.OnException event. If you need to use this event in your code, use the global ApplicationException variable instead; the component will forward all exceptions there.

11. *Chartboost, Facebook and AdMob*

File: JVE.ChartBoost.pas Inheritance: TComponent ← TJVEChartBoost

File: JVE.AdMob.pas Inheritance: TComponent ← TJVEAdMobInterstitial

File: JVE.Facebook.pas Inheritance: TComponent ← TJVEFacebookInterstitial

These components provide interfaces for Chartboost, Facebook Audience Network and AdMob SDKs for presenting interstitial or full screen ads within a mobile application.

To present Chartboost ads you will need to register at https://www.chartboost.com.

To present AdMob ads you will need to register at http://www.admob.com.

To present Facebook ads, register at https://www.facebook.com/audiencenetwork.

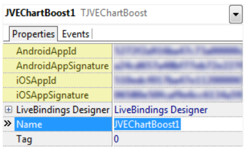
Each of the platforms above requires you to separately register the app, for each supported platform (currently this suite only support iOS and Android for advertising) and each provides one or two IDs for the app.

The platform is specified in a prefix "iOS" or "Android" of the name of the property (except for AdMob, which requires separate keys for cell and tablet). The properties thus defined are:

SDK	SDK ID	iOS Property Name	Android Property Name
Chartboost implemented by `TJVEChartBoost`	App ID	`iOSAppId`	`AndroidAppId`
	App Signature	`iOSAppSignature`	`AndroidAppSignature`
AdMob implemented by `TJVEAdMobInterstitial`	Unit Id	`iPhoneUnitId`	`AndroidUnitId`
		`iPadUnitId`	
Facebook implemented by `TJVEFacebook Interstitial`	Placement Id	`iPhonePlacementId`	`AndroidPlacementId`
		`iPadPlacementId`	

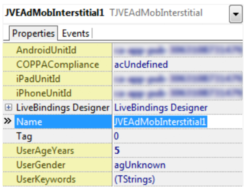
The list of properties, exposed by each component, is:

`TJVEChartBoost`



AndroidAppId	[redacted]
AndroidAppSignature	[redacted]
iOSAppId	[redacted]
iOSAppSignature	[redacted]
⊞ LiveBindings Designer	LiveBindings Designer
» Name	JVEChartBoost1
Tag	0

`TJVEAdMob Interstitial`



AndroidUnitId	[redacted]
COPPACompliance	acUndefined
iPadUnitId	[redacted]
iPhoneUnitId	[redacted]
⊞ LiveBindings Designer	LiveBindings Designer
» Name	JVEAdMobInterstitial1
Tag	0
UserAgeYears	5
UserGender	agUnknown
UserKeywords	(TStrings)

Additional properties are available only for some of the networks. **AdMob** allows you to provide optional user information (providing these values might improve the quality of ads, effectively resulting in a higher level of revenues), as follows:

- `UserGender` – if (expected) gender of the user is known, set it here.
- `UserAgeYears` – if the (expected) age of the user is known, use this property to set it.
- `UserKeywords` – list any known user's interests in this property.
- `UserLocation` – if the current location of the user is known, it can be provided in this public (not published) property.
- `COPPACompliance` – indicates that the SDK should be compliant with COPPA act.

All ad networks have these additional properties and events:

- `Reward` – indicates that Reward Video should be presented instead of a regular interstitial.
- `OnReward` – event, indicating that the user watched the video and is eligible for reward.
- `OnAdShown` – event, indicating that an interstitial started presenting (this event is not usually needed, as a cached ad is presented immediately; this event is not available for Chartboost).
- `OnAdClosed` – event, indicating that the interstitial was dismissed.
- `OnNoAd` – event, indicating that caching has failed and there is no appropriate ad to show.

AdMob component, is a singleton. You cannot have multiple components, caching and presenting different ads (you can have two: one for interstitials and one for reward videos, but that is the limit).

Some further events are available for some of the interstitial networks, which are not mandatory for implementation.

The following methods are exposed by all the components:

- `Cache` – caches the ad from the servers into the device.
- `IsCached` – returns True if the ad was successfully cached.
- `Show` – presents the ad. If the ad was cached, presents it immediately, otherwise caches it and presents it. If there are no ads currently available – does nothing.

See *Interstitials* support section below for more use cases for these components.

It should be noted that these components are only available in mobile applications (since the underlying SDKs are only available there), so these components do nothing on other platforms (`Cache` and `Show` perform no action, while `IsCached` always returns `False`).

Example ad, delivered by Chartboost:

12. Banners and Interstitials

This component suite provides components, which present banners and interstitials. They can be used for cross promotions and in-house ads or they can be used as a basis for paid ads services.

For banner paid ads, see `TJVEAdMob` control (*AdMob Support and Mediation* section below), which subclasses from `TJVEBanners` and uses its capabilities as a fallback mechanism, when ads are not available.

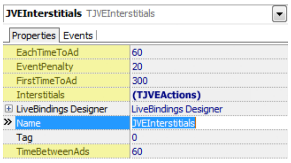
For a better way to present paid interstitials, see *Interstitials Use Case* section below.

12.1 Presenting Interstitials

File: `JVE.Interstitials.pas` Inheritance: `TComponent` ← `TJVEInterstitials`

This control presents interstitials (ads and promos) on major events. For example, finishing a level is a major event in your game; you can use this component to show ads / promos when the user finishes a level. The properties exposed by this component are:

- `Interstitials` – the list of interstitials to show. See *List of Ads / Cross Promos* section below for details.



JVEInterstitials TJVEInterstitials

EachTimeToAd	60
EventPenalty	20
FirstTimeToAd	300
Interstitials	(TJVEActions)
⊞ LiveBindings Designer	LiveBindings Designer
Name	JVEInterstitials
Tag	0
TimeBetweenAds	60

- `EachTimeToAd` – time (in seconds) to pass from the moment the application is started, until the interstitial is presented the first time.
- `FirstTimeToAd` – time to pass from the moment the application is started **for the very first time**, until the interstitial is presented the first time. It is generally advisable to limit the number of ads the user sees the very first time the app is opened.
- `TimeBetweenAds` – time to pass between interstitials presentations.
- `EventPenalty` – if events occur more often than usual, you can specify the penalty time (in seconds) for each major event.

This component does not expose any events and just one function: `ProcessMajorEvent`. Use this function to mark the major event within the app. This function is the one, which could show the interstitial (if appropriate).

It will return True, if the interstitial was (and thus, is being) presented (the function returns immediately).

Since clicking the interstitial might take the user out of the app and it is not known when (if) he will be back, if this function returns True, make sure the app is idle (waiting for user input) and its state is saved.

An interstitial is presented on major events (calls to the `ProcessMajorEvent` function), but not before a specified time has passed.

You can specify a given time between ads (or 0, in case every major event deserves an ad). You can also specify a time penalty for major events, so you can specify:

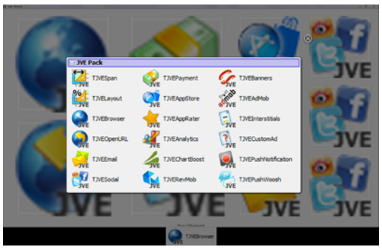
- Time Between Ads – 60 seconds
- Major Event Penalty – 20 seconds

A usual user will see an ad every 60 seconds or more, but a user, which opens a level and closes it immediately will see an ad sooner: maybe after just 20 seconds, if in that time he managed to open and close levels 3 times.

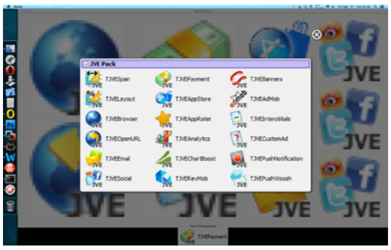
In addition to the above functionality, this component also exposes one class procedure: `ShowInterstitial(Bitmap, ClickCallback, CloseCallback)`. This procedure is provided as a convenience. It presents an interstitial and calls the call back procedure, if the user clicked the interstitial.

The presentation of the interstitials generally looks the same on all platforms:

MS Windows



Mac OS X

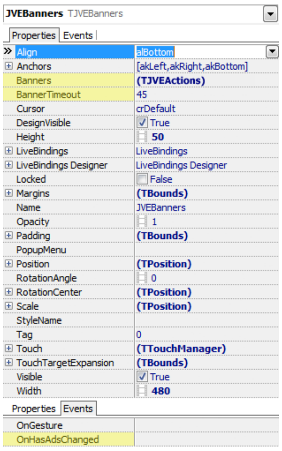


iOS Device and Simulator

12.2 Presenting Banners

File: `JVE.Banners.pas` Inheritance: `TControl` ← `TJVEBanners`

This control presents an ad-banner like control, which cycles through the defined banners. The best placement for the banners control is aligned to the bottom of the application window. The following properties are exposed by this control:



JVEBanners TJVEBanners

Properties | Events

Align	alBottom
Anchors	[akLeft,akRight,akBottom]
Banners	(TJVEActions)
BannerTimeout	45
Cursor	crDefault
DesignVisible	☑ True
Height	50
LiveBindings	LiveBindings
LiveBindings Designer	LiveBindings Designer
Locked	□ False
Margins	(TBounds)
Name	JVEBanners
Opacity	1
Padding	(TBounds)
PopupMenu	
Position	(TPosition)
RotationAngle	0
RotationCenter	(TPosition)
Scale	(TPosition)
StyleName	
Tag	0
Touch	(TTouchManager)
TouchTargetExpansion	(TBounds)
Visible	☑ True
Width	480

Properties | Events

OnGesture	
OnHasAdsChanged	

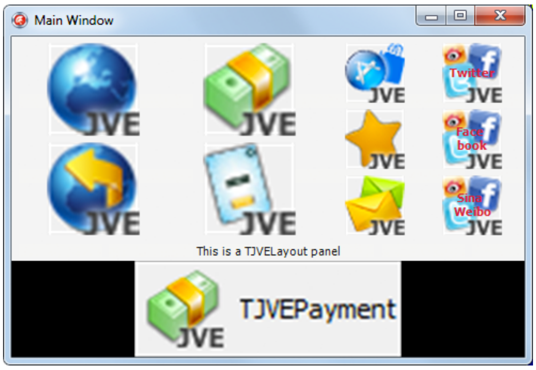
- `Banners` – the list of banners to show. See *List of Ads / Cross Promos* section below for details.
- `BannerTimeout` – the time (in seconds) each banner is displayed, before the control switches to the next available banner.

In addition to these, a single event is provided by this control: `OnHasAdsChanged`. This event is called when a banner becomes available or when no banners are now available (a read/only `HasAds` property is also available, returning the current control state).

You can use this event to hide the control and reclaim its space in the circumstances when you might be in a situation where you no longer have any ads, which might be presented.

This control can be used as a fallback for actual paid ads service. You can use a ready `TJVEAdMob` control for that or, if you prefer another ad network, see `TJVEAdMob` for an example implementation of this sort.

The banners are presented as images stretched (but preserving aspect ratio) on the black background. Here is MS Windows presentation example:



12.3 List of Ads / Cross Promos

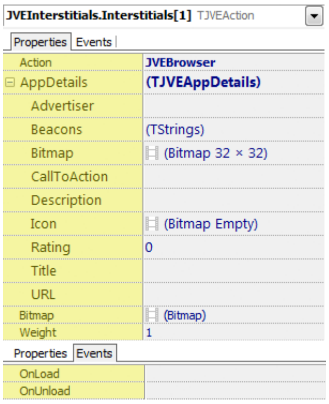
Both, `TJVEInterstitials` and `TJVEBanners` use a collection of `TJVEActions` to hold the banners / interstitials, which can be included in the ads rotation.

The following properties are exposed by these collection items (no functions are exposed by them):

- `Action` – specifies the action, to be taken if the ad is clicked. This should be a component implementing `IJVEExecutable` or `IJVEInterstitial` interface. See below for details.
- `Weight` – the relative frequency, this ad should be shown, relative to other ads.
- `Bitmap` – the banner bitmap that should be presented for this ad. Only exposed for `IJVEExecutable`. Only present for backward compatibility and/or interstitial-only actions. See `AppDetails` below.
- `Identifier` – the identifier, which should be paid for (only for `TJVEPayment`).
- `AppDetails` – exposes the ad details: title, description, advertiser, rating, icon, banner (the legacy Bitmap property) and call to action to use. It also includes URL (click target) and list of beacons (URL to call when the ad is presented). Only exposed for `IJVEExecutable`.

The `TJVEAction` items have design-time editors, which show and hide properties, based on the Action selection. So, based on your selection you might see any of the following:

`IJVEExecutable`

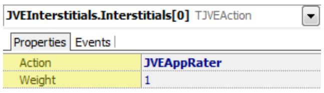


JVEInterstitials.Interstitials[1] TJVEAction

Action	**JVEBrowser**
⊟ AppDetails	**(TJVEAppDetails)**
Advertiser	
Beacons	(TStrings)
Bitmap	(Bitmap 32 × 32)
CallToAction	
Description	
Icon	(Bitmap Empty)
Rating	0
Title	
URL	
Bitmap	(Bitmap)
Weight	1

OnLoad	
OnUnload	

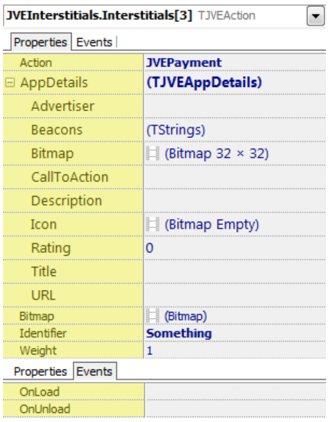
`IJVEInterstitial`



JVEInterstitials.Interstitials[0] TJVEAction

Action	**JVEAppRater**
Weight	1

`TJVEPayment`



JVEInterstitials.Interstitials[3] TJVEAction

Action	**JVEPayment**
⊟ AppDetails	**(TJVEAppDetails)**
Advertiser	
Beacons	(TStrings)
Bitmap	(Bitmap 32 × 32)
CallToAction	
Description	
Icon	(Bitmap Empty)
Rating	0
Title	
URL	
Bitmap	(Bitmap)
Identifier	**Something**
Weight	1

OnLoad	
OnUnload	

A design-time editor is provided for the `Action` property itself: it will only list components, implementing those two interfaces. If you assign an invalid component to the `Action` property in runtime, that action will be skipped from the rotation.

The `Identifier` property will allow you to select one of the payment item identifiers, defined within the selected `TJVEPayment`. Again, if invalid identifier is selected indirectly, the action will be excluded from the rotation.

`IJVEInterstitial` entities provide their own user interface, so there is no `Bitmap` property for them and they cannot be selected for rotation in the `TJVEBanners` control.

If `Bitmap` or `AppDetails` property is in effect, you can also provide `OnLoad` and `OnUnload` events, which could be used to load and unload the bitmaps on demand to preserve memory (especially important on mobile devices).

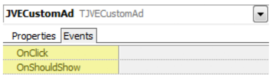
The actions will be automatically excluded from rotation, if they are not currently valid: `IJVEInterstitial` will be excluded if it is not yet cached; `IJVEExecutable` will be excluded if it cannot be opened (for example if a persistent payment was already made or if a cross-promoted app is already installed and the `ExistsAction` is set to ignore).

12.4 Custom Ads

You can create your own implementations of `IJVEInterstitial` (see `TJVEChartBoost` for a convenient example) or of `IJVEExecutable` (see `TJVEOpenURL` for a convenient example). If you don't want to subclass, a basic implementation of `IJVEExecutable` is provided as part of this suite:

File: `JVE.Interstitials.pas` Inheritance: `TComponent` ← `TJVECustomAd`

This component does not expose any properties or methods at all, but it exposes two events, which are defined as follows:



- `OnClick` – occurs when the user clicks on the presented ad.
- `OnShouldOpen` – occurs just before the final decision to present this ad is taken. Return `False` value in the `Show` parameter, if you don't want this ad to be presented now (for any reason). If you don't provide this event, the ad is always considered to be valid.

If you need to update the bitmap and any other aspect of the ad, which will be presented to the user, you can also do that within the `OnShouldOpen` event (it is directly accessible there as `Item.Bitmap` or `Item.AppDetails`).

12.5 Interstitials Use Case

Here is the way, to define the `Interstitials` collection of the `TJVEInterstitials` component, if you want to present Chartboost ads, have a fallback to Facebook Audience Network and then to cross-promo of your other two apps and have an app rating reminder:

Action	Weight	Comments
`TJVEChartBoost` component	10000	
`TJVEFacebookInterstitial`	100	
`TJVEAppStore` component	1	
`TJVEAppStore` component	1	
`TJVEAppRater` component	1	This component gets an artificial weight boost, as it is to be shown once every several days, so its weight is of no consequence.

Notice, you would normally have several fallback ads, not to bore your user too much.

13. AdMob and Facebook Audience Network

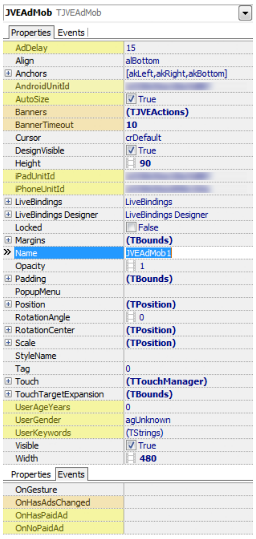
File: `JVE.AdMob.pas` Inheritance: `TControl` ← `TJVEBanners` ← `TJVEAdMob`

File: `JVE.Facebook.pas` Inheritance: `TControl` ← `TJVEBanners` ← `TJVEFacebookBanner`

These controls can integrate the Google AdMob SDK or Facebook Audience Network into your application. AdMob SDK can present paid AdMob and other ads (ads by other vendors are presented through its mediation service); see http://www.admob.com site for details. Audience Network can present paid Facebook ads; see https://www.facebook.com/audiencenetwork.

AdMob and Facebook SDKs are only available on iOS and Android platforms; these controls inherits from `TJVEBanners` thus when the SDK is not available (or when there are no ads to show), it will fallback to in-house ads and cross-promos, supplied by the `TJVEBanners` normal functionality.

This control inherits from TJVEBanners, so this section will not cover inherited properties (see *Banners and Interstitials* section, *Presenting Banners* subsection, instead). The following properties are exposed by this control:



JVEAdMob TJVEAdMob

Properties | Events

AdDelay	15
Align	alBottom
⊞ Anchors	[akLeft,akRight,akBottom]
AndroidUnitId	[redacted]
AutoSize	☑ True
Banners	(TJVEActions)
BannerTimeout	10
Cursor	crDefault
DesignVisible	☑ True
Height	90
iPadUnitId	[redacted]
iPhoneUnitId	[redacted]
⊞ LiveBindings	LiveBindings
⊞ LiveBindings Designer	LiveBindings Designer
Locked	☐ False
⊞ Margins	(TBounds)
» Name	JVEAdMob1
Opacity	1
⊞ Padding	(TBounds)
PopupMenu	
⊞ Position	(TPosition)
RotationAngle	0
⊞ RotationCenter	(TPosition)
⊞ Scale	(TPosition)
StyleName	
Tag	0
⊞ Touch	(TTouchManager)
⊞ TouchTargetExpansion	(TBounds)
UserAgeYears	0
UserGender	agUnknown
UserKeywords	(TStrings)
Visible	☑ True
Width	480

Properties | Events

OnGesture	
OnHasAdsChanged	
OnHasPaidAd	
OnNoPaidAd	

- `AndroidUnitId`, `iPadUnitId` and `iPhoneUnitId` – the Ad Unit ID or Mediation ID, as provided by the AdMob service. Due to AdMob requirements you should have separate iPad and iPhone IDs (the control will automatically choose the correct one for you).
- AndroidPlacementId, iPadPlacementId and iPhonePlacementId – the ad Placement ID, as provided by the Audience Network service.
- `AdDelay` – the time before actual ads are requested from the AdMob server (if fallback banners are available).

The following properties are only available for AdMob ads:

- `AutoSize` – AdMob SDK prescribes a particular height for this control. Set to `True` to have the control set its height as needed.
- `UserGender` – providing (expected) user details generally allows for better ad matches, resulting in a higher click-through-rate and, in the end, higher revenues. If gender is known, set it here.
- `UserAgeYears` – set this field to the estimated age of the user.
- `UserKeywords` – any currently valid keywords, identifying the app and/or your user can be set here.
- `UserLocation` – if the current location of the user is known, it can be provided in this public (not published) property.

This control provides no new methods, but it exposes two further events (beyond the one, exposed by `TJVEBanners`), as follows:

- `OnHasPaidAd` – this event is called the moment an ad is successfully retrieved from the server and shown.
- `OnNoPaidAd` – this event is called when there are no ads available (for example if there is no valid ads for the user or if there is no internet connectivity). An error message, returned by the SDK, is passed to this event for reference. You should not normally present this error.

You can use the above two events, if you don't want to use the fallback banners. You can then show the control in the first event and hide it in the second (and reclaim the screen real-estate appropriately).

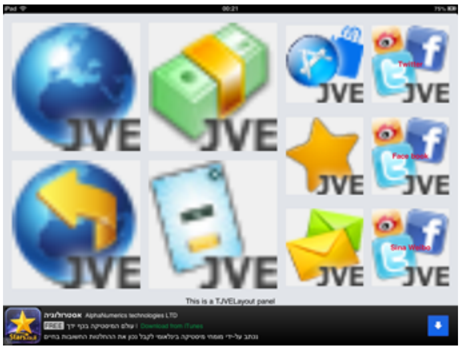
It usually takes a couple of seconds for the component to bring the very first ad, so if `AdDelay` value is 0 and there is a fallback banner, that fallback might be presented for just 2 seconds: too short to react and too distracting.

Since this control normally sets its own height, its best placement is aligned to bottom on your form, with the remaining user interface within a separate TLayout, aligned to client. See control's source code's comments for details on the exact height that will be used.

AdMob SDK support showing other ad networks' ads through a free mediation mechanism (see AdMob website for details). You can install other mediation layers as needed (see `JVE.AdMob.pas` file for details and explanations).

To configure AdMob mediation (i.e. specify which networks' ads will be presented with what frequencies and where) you will have to go to the AdMob website. These issues are beyond what this component attempts to expose.

Example presentation of an actual ad, delivered by the AdMob service, on iPad device, might look as follows:


iPad
00:21
JVE
Twitter
Face book
Sina Weibo
This is a TJVELayout panel
אסטרולוגיה AlphaNumerics technologies LTD
FREE עולם המיסטיקה בכף ידך ! Download from iTunes
נכתב על-ידי מומחי מיסטיקה בינלאומי לקבל נכון את ההחלטות החשובות בחיים

14. *PubNative and MobFox*

File: `JVE.PubNative.pas` Inheritance: `TComponent` ← `TJVEPubNative`

File: `JVE.MobFox.pas` Inheritance: `TComponent` ← `TJVEMobFox`

These components provide interfaces for PubNative and MobFox APIs for presenting native ads within a mobile application.

To present PubNative ads you will need to register at `https://www.pubnative.net`.

To present MobFox ads you will need to register at `http://www.mobfox.com`.

Each of the platforms above requires you to separately register the app, for each supported platform (currently this suite only support iOS and Android for advertising) and each provides a single ID for the app for each platform.

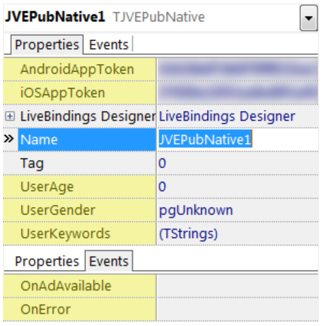
The platform is specified in a prefix "iOS" or "Android" of the name of the property. The properties thus defined are:

API	API ID	iOS Property Name	Android Property Name
PubNative implemented by `TJVEPubNative`	App Token	`iOSAppToken`	`AndroidAppToken`
MobFox implemented by `TJVEMobFox`	Inventory Hash	`iOSInventoryHash`	`AndroidInventoryHash`

MobFox API features a demo key, which is supplied as the default value for the component; it could be used to test your app layout and integration. Notice also that you will not be able to test integration with your own key (i.e. inventory hash) before MobFox approves your app.

The list of properties, exposed by each component, is:

`TJVEPubNative`



JVEPubNative1 TJVEPubNative

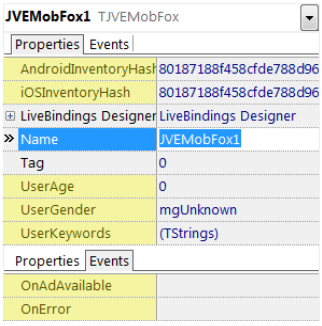
Properties | Events

AndroidAppToken	[redacted]
iOSAppToken	[redacted]
⊞ LiveBindings Designer	LiveBindings Designer
» Name	JVEPubNative1
Tag	0
UserAge	0
UserGender	pgUnknown
UserKeywords	(TStrings)

Properties | Events

OnAdAvailable	
OnError	

`TJVEMobFox`



JVEMobFox1 TJVEMobFox

Properties | Events

AndroidInventoryHash	80187188f458cfde788d96
iOSInventoryHash	80187188f458cfde788d96
⊞ LiveBindings Designer	LiveBindings Designer
» Name	JVEMobFox1
Tag	0
UserAge	0
UserGender	mgUnknown
UserKeywords	(TStrings)

Properties | Events

OnAdAvailable	
OnError	

Both APIs are extremely similar in the parameters they accept and expose. The following properties are available in both components:

- `UserGender` – if (expected) gender of the user is known, set it here.
- `UserAge` – if the (expected) age of the user is known, use this property to set it.
- `UserKeywords` – list any known user's interests in this property.
- `UserLocation` – if the current location of the user is known, it can be provided in this public (not published) property.

All these properties are optional, but providing their values might improve the quality of ads, effectively resulting in a higher level of revenues.

The following events are exposed by these components:

- `OnAdAvailable` – called when ad information was successfully retrieved from the server.
- `OnError` – called if an ad cannot be retrieved. You would usually log this error but not present it (unless showing the ad is mandatory n your app).

A single method is exposed by both components: `RequestAd([Done])`. When called without a parameter it starts the ad request; upon the request completion it calls either `OnAdAvailable` or `OnError` event (depending on the result).

If called with a callback parameter, the callback is called instead of the `OnAdAvailable` event. In case of failure, when calling with a callback, both the `OnError` event will be called and the callback will be called with `nil` instead of the ad data.

The ad data is represented by the `TJVEAppDetails` class (see *List of Ads / Cross Promos* section above), containing the following properties:

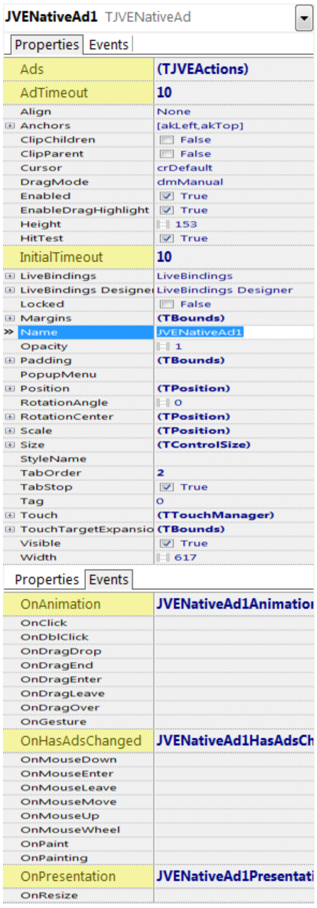
- `Title`, `Description`, `Advertiser` – textual description fields.
- `CallToAction` – a text for the button to open the ad (usually “Download” or “Install”).
- Rating – number of stars of rating (1-5 scale).
- `Icon` and `Bitmap` – an icon (square) and a banner (1:1.91 aspect ratio) of the ad.
- `URL` – the address to open when the user clicked the ad.
- `Beacons` – an array of addresses, which should be called, after the ad is presented.

It should be noted that these components are only available in mobile applications, so these components do nothing on other platforms (`RequestAd` simply calls `OnError`).

15. Native Ads Presentation

File: `JVE.Native.pas` Inheritance: `TControl` ← `TJVENativeAd`

This component suite provides controls to conveniently present native ads. They can be used to present MobFox or PubNative ads or cross promotions and in-house ads.



JVENativeAd1 TJVENativeAd

Properties | Events

Ads	(TJVEActions)
AdTimeout	10
Align	None
⊞ Anchors	[akLeft,akTop]
ClipChildren	☐ False
ClipParent	☐ False
Cursor	crDefault
DragMode	dmManual
Enabled	☑ True
EnableDragHighlight	☑ True
Height	153
HitTest	☑ True
InitialTimeout	10
⊞ LiveBindings	LiveBindings
⊞ LiveBindings Designer	LiveBindings Designer
Locked	☐ False
⊞ Margins	(TBounds)
Name	JVENativeAd1
Opacity	1
⊞ Padding	(TBounds)
PopupMenu	
⊞ Position	(TPosition)
RotationAngle	0
⊞ RotationCenter	(TPosition)
⊞ Scale	(TPosition)
⊞ Size	(TControlSize)
StyleName	
TabOrder	2
TabStop	☑ True
Tag	0
⊞ Touch	(TTouchManager)
⊞ TouchTargetExpansio	(TBounds)
Visible	☑ True
Width	617

Properties | Events

OnAnimation	JVENativeAd1Animatio
OnClick	
OnDblClick	
OnDragDrop	
OnDragEnd	
OnDragEnter	
OnDragLeave	
OnDragOver	
OnGesture	
OnHasAdsChanged	JVENativeAd1HasAdsCh
OnMouseDown	
OnMouseEnter	
OnMouseLeave	
OnMouseMove	
OnMouseUp	
OnMouseWheel	
OnPaint	
OnPainting	
OnPresentation	JVENativeAd1Presentati
OnResize	

The first control presents the container for the native ad. It manages the ad cycling and clicking behavior (but not the content; see *Presenting Ad Content* section below). The following properties are exposed by this control:

- `Ads` – the list of ads to show. See *List of Ads / Cross Promos* section above for details (part of the *Banners and Interstitials* documentation).
- `InitialTimeout` – indicates the time (in seconds) to present the first ad (usually an in-house fallback, before the paid ad is downloaded). Zero value stops the timer.
- `AdTimeout` – the time (in seconds) each banner is displayed, before the control switches to the next available ad.
- `HasAds` – read/only public (not published) property, indicating whether an ad is being presented.

The following events are exposed by this control:

- `OnAnimation` – called when the new ad is available and is about to be presented; see *Animating Ad Switches* section below for details.
- `OnHasAdsChanged` – called when the ad becomes available after there was none and vice versa. Can be used to reorganize the screen to compensate for the missing ad.
- `OnPresentation` – called when a new ad is available; could be used to update the user interface, for example, based on the amount of text in textual fields, etc.

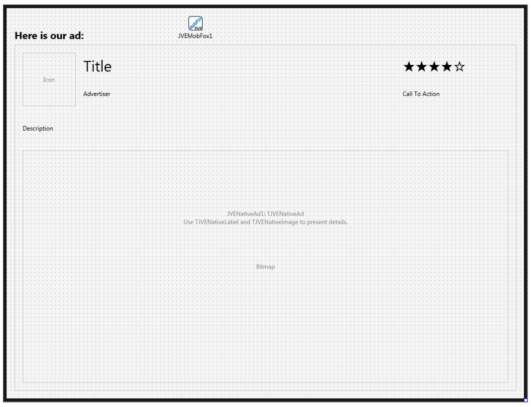
Notice, the `OnClick` event will only be called if an ad is actually presented; use `OnMouseDown` to be called regardless of whether the ad is presented.

In addition to the above, the following methods are provided by this control:

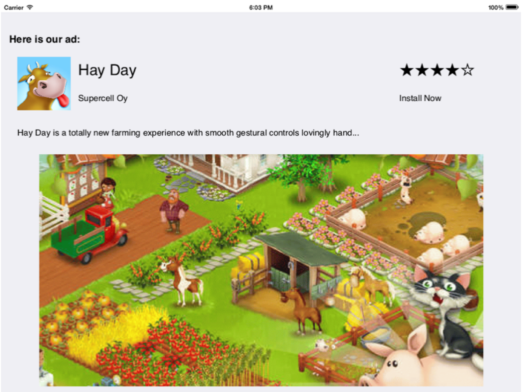
- `NextAd` – attempts to present the next ad immediately, without waiting for the timeout.
- `AdClicked` – continues as if the user clicked the ad. See *Implementing Parental Gates* section for a use case.
- `UpdateVisualControl` – this class procedure reapplies the ad to the presentation controls therein. Used to implement custom native ads presentation; see `TJVENativeLabel/Image` for an implementation example.

Several native ad controls can be linked and synchronized to always show the same ad. To achieve this set one of the native ad control's `Ads` property to a proper list of ads to show, while in the other control add the first native ad control as the only ads provider.

The presentation of the interstitials is generally identical on all platforms and precisely follows your design-time definitions, as can be seen here:


Here is our ad:
JVEMobFox1
Title
★★★★☆
Icon
Advertiser
Call To Action
Description
JVENativeAd1: TJVENativeAd
Use TJVENativeLabel and TJVENativeImage to present details.
Bitmap

→



Hay Day is a totally new farming experience with smooth gestural controls lovingly hand...

15.1 Presenting Ad Content

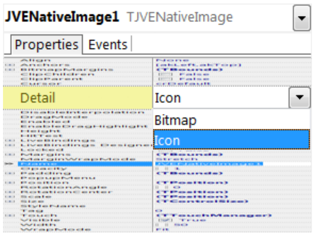
File: `JVE.Native.pas` Inheritance: `TImage` ← `TJVENativeImage`

File: `JVE.Native.pas` Inheritance: `TLabel` ← `TJVENativeLabel`

Two controls are provided to present the ad content (icon, banner and textual fields). If you place these controls anywhere within the `TJVENativeAd`, they will present the appropriate value from the ad and be updated automatically (do not place them in a standalone fashion: they will perform no action there).

Both controls hide their regular data properties (i.e. `Text`, `Bitmap` and `MultiResBitmap`), but they expose a single new property: `Detail`. This property indicates which piece of ad content is presented by the control:

Icons and banners:



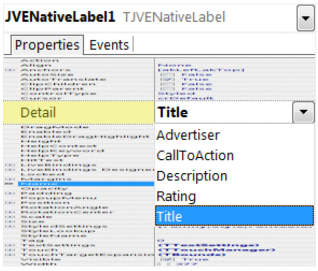
JVENativeImage1 TJVENativeImage

Properties | Events

Detail: Icon

Bitmap
Icon

Textual fields:



JVENativeLabel1 TJVENativeLabel

Properties | Events

Detail: Title

Advertiser
CallToAction
Description
Rating
Title

If for your implementation these two controls don't provide sufficient presentation capabilities (for example, if you want custom stars presentation for Rating), you can create your own controls, which can support automatic ad data updates by implementing the `IJVENativeAdDetail` interface.

15.2 Implementing Parental Gates

If you wish to implement a parental gate on native ads, override the `OnClick` event and perform the following:

- Disable the ad timer (i.e. `AdTimeout:=0`).
- Present the parental gate.
- If confirmed, call the `AdClicked` method to simulate the click.
- Restart the ad time (i.e. `AdTimeout:=45`).

15.3 Animating Ad Switches

To animate the switches between ads, the following should be implemented:

- Add the transition effect you want to the `TJVENativeAd`. You can disable it for design-time convenience.
- Add a float animation on the Progress property of the transition effect. Set `StartValue` to 100, `StopValue` to 0 and `Duration` to the transition speed you want.
- Implement the `TJVENativeAd`'s `OnAnimation` event as follows:

```
begin
  Transition.Enabled := True; // If you have disabled it previously
  Transition.Target.Assign(Source); // We are reversing the animation
  Animation.Start;
end;
```

This implementation will allow you to use any of the Delphi provided transition effects to perform the switch.

16. *Push Notifications*

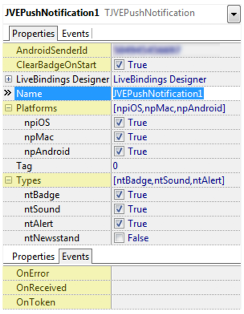
File: `JVE.Pushes.pas` Inheritance: `TComponent` ← `TJVEPushNotification`

This component implements the Push Notifications support, with regard to the Operating System integration. Push notifications support, provided below, does not work in Windows or in iOS Simulator (i.e. only Android and iOS Devices and Mac OS X 10.7 or later).

Push notifications are singleton by design. Care should be taken to only ever have one instance of `TJVEPushNotification` (or any descendant thereof) ever created within the app.

This class provides operating system integration, but not the communication with your server-side (for a complete implementation, which could also be used as an example of server integration, see *PushWoosh Service Integration* section below). This component exposes the following properties:

- `AndroidSenderId` – GCM Sender ID, assigned for the application.
- `Platforms` – the list of platforms, on which this component is operational.
- `Types` – the list of notifications, for which the component is registering.
- `ClearBadgeOnStart` – setting this to True will instruct the application to clear the app icon badge at the start of the app.



JVEPushNotification1 TJVEPushNotification

Properties | Events

AndroidSenderId	[redacted]
ClearBadgeOnStart	☑ True
⊞ LiveBindings Designer	LiveBindings Designer
» Name	JVEPushNotification1
⊟ Platforms	[npiOS,npMac,npAndroid]
npiOS	☑ True
npMac	☑ True
npAndroid	☑ True
Tag	0
⊟ Types	[ntBadge,ntSound,ntAlert]
ntBadge	☑ True
ntSound	☑ True
ntAlert	☑ True
ntNewsstand	□ False

Properties | Events

OnError	
OnReceived	
OnToken	

Additional functionality provided by the component is the ability to convert Apple's APN token to Google's FCM token (for iOS). To enable this, set the following properties:

- `FCMConversionApiKey` – Server API key for the Google project.
- `FCMConversionIsSandbox` – True will use the sandbox account; False – production.

Leave `FCMConversionApiKey` property empty, if you want to receive Apple token as is. Notice, if you receive error `-1012`, this means that the Api Key is invalid or does not match the Bundle Identifier of the application.

This component exposes no methods. The following events are available:

- `OnToken` – this event is invoked once a push notification token is available for your app. Notice, the fact that you get the token does not mean that the user agreed to accept them.
- `OnError` – this event is invoked, if the operating system failed to get a valid token for the app. An error message is provided as a parameter; the default behavior is to present it.
- `OnReceived` – this event is called for each push notification, received from the server. The push notification details are provided within the `Push` parameter (see below); `OnAppStart` indicates whether the app was started to take care of this notification (rather than receiving the notification while the app was already running). The default behavior is to present the notification message in a standard message box.

The `Push` parameter sent to the `OnReceived` event is actually a `TStrings` descendant. You can use the standard `Values` property to access the content of the notification. This descendant provides the following functions to access standard notification details:

- `GetMessage` – returns the notification text.
- `GetBadge` – returns the notification requested badge number.
- `GetSound` – returns the sound to play along the notification.
- `GetNewsstand` – returns `True`, if this is a newsstand content availability notification.

The format of the string list is based on the original Apple generated dictionary, converted using the *Apple Dictionary to String List Converter* (see *Other Utilities* section below).

Further implementations for specific services can introduce class helpers to provide more standard readers (like the *PushWoosh Service Integration* does).

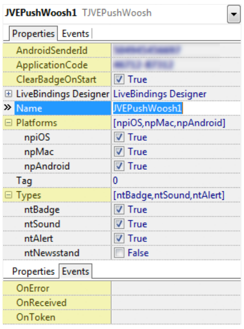
If you have problems making push notifications work, examine the `JVE.Pushes.pas` file header, which contains a list of steps, you should take to make them work.

17. *PushWoosh Service Integration*

File: `JVE.PushWoosh.pas` Inheritance: `TComponent` ← `TJVEPushNotification` ← `TJVEPushWoosh`

As mentioned above, the `TJVEPushNotification` component only takes care about the operating system integration, not communications with any push notifications provider server.

The `TJVEPushWoosh` component provides an interface to the PushWoosh service for push notifications delivery; see http://www.pushwoosh.com for more details. This component takes care of all the required communications with this server.



JVEPushWoosh1 TJVEPushWoosh

Properties | Events

AndroidSenderId	[redacted]
ApplicationCode	[redacted]
ClearBadgeOnStart	☑ True
⊞ LiveBindings Designer	LiveBindings Designer
» Name	JVEPushWoosh1
⊟ Platforms	[npiOS,npMac,npAndroid]
npiOS	☑ True
npMac	☑ True
npAndroid	☑ True
Tag	0
⊟ Types	[ntBadge,ntSound,ntAlert]
ntBadge	☑ True
ntSound	☑ True
ntAlert	☑ True
ntNewsstand	☐ False

Properties | Events

OnError	
OnReceived	
OnToken	

A single new mandatory property is exposed by this component: `ApplicationCode`, which is the identifier, assigned to your application by the PushWoosh service.

In addition to this, the Push content class provides the following functions, defined in a helper class:

- `GetLink` – returns the link, which was sent with the message.
- `GetHash` – returns the message's unique identifier (should not normally be used).
- `GetCustomData` – returns any custom data sent with the notification.

It should be noted that this implementation of the PushWoosh interface specifically does **not** support HTML Pages notifications. Please do not attempt to send these to your Delphi app.

PushWoosh service also supports sending user's location and various user-related tags (which could all be used then to filter the user, which should receive a particular notification). To use these services you might need a premium PushWoosh account.

Access to these services is provided via a number of public (not published) properties:

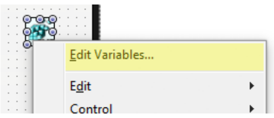
- `Location` – the current world location of the user.
- `TagString[Name]:String` – string tag value; use to get or set.
- `TagInteger[Name]:Integer` – integer tag value; use to get or set.
- `TagList[Name]:TArray<String>` – list tag value; use to get or set.
- `DeleteTag(Name)` – this is a procedure, not a property; call this to delete a tag.

Notice, PushWoosh preserves current values on its services, so the client implementation mimics that, keeping all the tags' values persistent in the device.

18. *Persistent Storage*

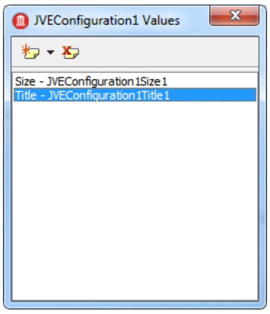
File: `JVE.Configuration.pas` Inheritance: `TComponent` ← `TJVEConfiguration`

This component allows you to easily store any configuration (hence the name) or other settings, which should be persisted between sessions. This component has no properties or events of its own; instead it provides a property editor allowing you to define multiple persistent properties:



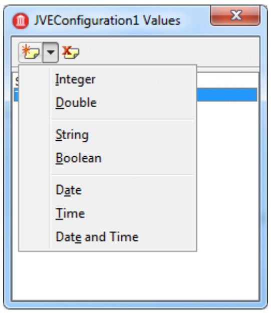
Via right-click or double-click in the designer.

→

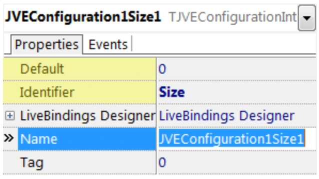


Size - JVEConfiguration1Size1
Title - JVEConfiguration1Title1

→



Every value you create belongs to the `TJVEConfiguration` component, but is also created directly in your code, allowing you type-safe, compile-time verified, access to all the values, as follows: `JVEConfiguration1Size1.Value` (for the value in the screenshot below).



Default	0
Identifier	Size
⊞ LiveBindings Designer	LiveBindings Designer
Name	JVEConfiguration1Size1
Tag	0

Each value exposes a total of three properties, as follows:

- `Identifier` – the identifier to use to access the persistent storage for the variable. Upon a change of this property in the Object Inspector the Name and its source code reference will be reset.
- `Default` – the default for the value, before another value is stored in the persistent storage. The type of this property depends on the type of the value created.
- `Value` (runtime only property) – access to this property is actually redirected to the persistent storage; use it to read or write the persistent value.

This component also provides the following class methods (they allow access in runtime, or if you don't want an overhead of individual value components):

- `WriteBool(Ident,Value)` – this and other `Write` functions preserve the value within the persistent storage with the given identifier.
- `WriteString(Ident,Value)`
- `WriteInteger(Ident,Value)`
- `WriteDate(Ident,Value)`
- `WriteDateTime(Ident,Value)`
- `WriteFloat(Ident,Value)`
- `WriteTime(Ident,Value)`
- `ReadBool(Ident,Default):Boolean` – this and other `Read` functions retrieve the value from the persistent storage with the given identifier. If no value is preserved therein, return the `Default` value.
- `ReadString(Ident,Default):String`
- `ReadInteger(Ident,Default): Integer`
- `ReadDate(Ident,Default): TDateTime`
- `ReadDateTime(Ident,Default): TDateTime`

- `ReadFloat(Ident,Default):Double`
- `ReadTime(Ident,Default):TDateTime`
- `DeleteKey(Ident)` – this function deletes a value from the persistent storage. Notice, I have seen this function fail sometimes on the Mac and iOS platforms, it is advisable not to use this function, instead writing the default value back into the storage.

This class uses Windows Registry on MS Windows, `JSharedPreferences` on Android and `NSUserDefaults` functionality on other platforms.

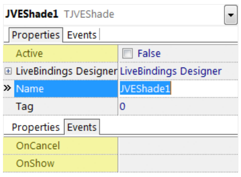
19. *In-Progress App Shading*

File: `JVE.Shade.pas` Inheritance: `TComponent` ← `TJVEShade`

This component allows you to easily shade the screen, while your application is contacting the server or performing another lengthy task.

It exposes a single Active property indicating whether the screen is shaded (changing this property shows or hides the shade using a fade animation). The component also exposes the following events:

- `OnShow` – this event is called when the shade is actually presented (could be used start heavy work in the main thread when shade is visible.
- `OnCancel` – occurs when the user clicks or taps the shade. Usually indicates that the user wants to cancel the operation.



JVEShade1 TJVEShade

Properties | Events

Active	□ False
⊞ LiveBindings Designer	LiveBindings Designer
» Name	JVEShade1
Tag	0

Properties | Events

OnCancel	
OnShow	

`OnCancel` event will not hide the shade automatically: it only notifies you of the user's wishes.

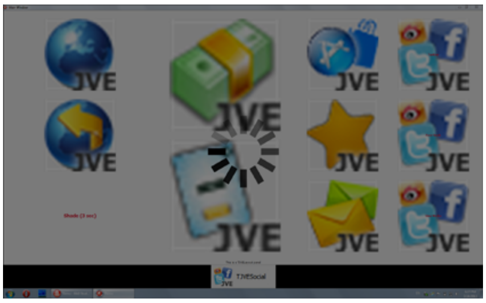
This component also provides two class methods:

- `Show([OnShow<Key>][, OnCancel])` – shades the screen; this function also returns the `Key` and passes it as a parameter to the OnShow event (if specified), which you should pass to the Hide function. The `OnShow` and `OnCancel` callbacks are same as the events.
- `Hide(Key)` – removes the shading. The shading is only removed when it was removed for all the keys, with which it was shown.

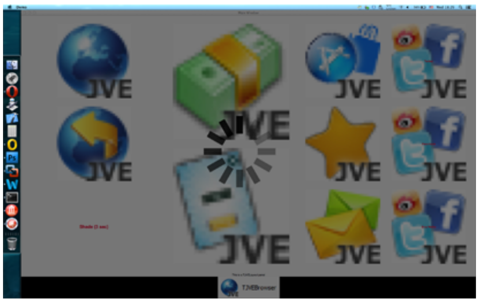
Such a `Key` implementation allows you to call the Show function several times and only when all shade requests are removed will the actual shade be hidden.

Notice, the screen will only be shaded as long as the operating system permits it. For example, if the iOS status bar is visible, it will not be shaded; if Mac OS X menu bar or Dock are visible, they will not be shaded, etc. Example shading presentations are:

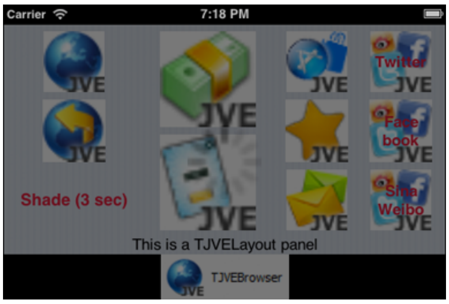
MS Windows



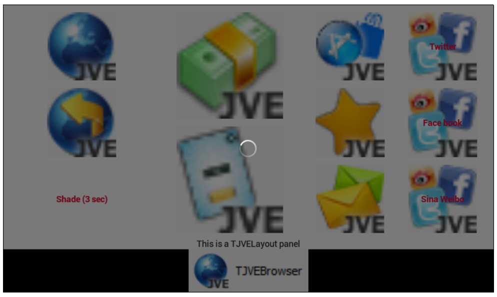
Mac OS X



iOS Device and Simulator



Android



20. Layout Controls

File: `JVE.Layout.pas` Inheritance: `TLayout` ← `TJVESpan` ← `TJVELayout`

These layout controls were created specifically to make dynamic size screens easier to develop. While FMX comes with a `TGridLayout`, it is really no match to the VCL `TGridPanel`.

`TJVELayout` works similarly to `TGridPanel`, with the following enhancements:

- Shares are used instead of percent. The difference: shares need not sum to 100% (for example, 2 shares out of 5 is 40%). You can create columns without worrying about the total number of shares, only keeping the proportions intact.
- A Column or a Row can have a combination of Shares and Pixels in its definition. First pixels are assigned to columns and rows, then remaining pixels are split amount the shares. This allows you to specify the minimum number of pixels for a column or a row.
- If you need to span multiple columns or rows, place `TJVESpan` (or even a child `TJVELayout`) on your `TJVELayout`, set `ColSpan` and/or `RowSpan` properties accordingly (same as in HTML).

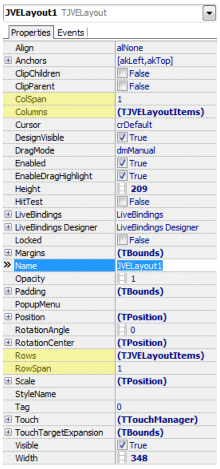
Controls placement, on the other hand, works similarly to the `TGridLayout`, except:

- A Column or a Row can be marked as a divider then child controls will skip this column or row.
- The `TJVELayout` control ensures that every column and every row start on an integer pixel and occupy integer number of pixels: not all controls look good, when there is a gap of half-a-pixel.
- All child controls of `TGridLayout` work as if having Align set to `alClient`; `TJVELayout` also support `alFit`, `alFitLeft` and `alFitRight`.

To ensure a perfect layout, especially with smaller columns and rows, `TJVELayout` will guarantee that all the columns, which have the same Share value, get exactly the same number of pixels per share.

In runtime neither control has any visual drawing. In design time the `TJVESpan` has a dashed line around it (same as most layout controls). The design time presentation of `TJVELayout` is more complicated: if will draw a red grid over columns and rows, marked as dividers and a green shade over the cells, which are open for placement.

The list of properties, this control exposed is (listed for `TJVELayout`; `TJVESpan` only exposes `ColSpan` and `RowSpan`):



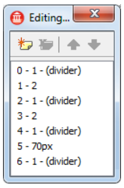
JVELayout1 TJVELayout

Properties | Events

Align	alNone
⊞ Anchors	[akLeft,akTop]
ClipChildren	□ False
ClipParent	□ False
ColSpan	1
Columns	(TJVELayoutItems)
Cursor	crDefault
DesignVisible	☑ True
DragMode	dmManual
Enabled	☑ True
EnableDragHighlight	☑ True
Height	209
HitTest	□ False
⊞ LiveBindings	LiveBindings
⊞ LiveBindings Designer	LiveBindings Designer
Locked	□ False
⊞ Margins	(TBounds)
» Name	JVELayout1
Opacity	1
⊞ Padding	(TBounds)
PopupMenu	
⊞ Position	(TPosition)
RotationAngle	0
⊞ RotationCenter	(TPosition)
Rows	(TJVELayoutItems)
RowSpan	1
⊞ Scale	(TPosition)
StyleName	
Tag	0
⊞ Touch	(TTouchManager)
⊞ TouchTargetExpansion	(TBounds)
Visible	☑ True
Width	348

Columns and Rows are collections

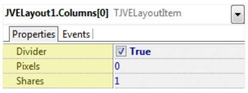
←



Editing...

0 - 1 - (divider)
1 - 2
2 - 1 - (divider)
3 - 2
4 - 1 - (divider)
5 - 70px
6 - 1 - (divider)

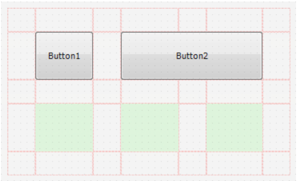
←



JVELayout1.Columns[0] TJVELayoutItem

Properties | Events

Divider	☑ True
Pixels	0
Shares	1



21. *Other Utilities*

File: JVE.Utils.pas

Several utilities, mostly used internally throughout the suite, are part of this file. Though they are internal, some of them might be useful, especially when developing for Mac OS X and iOS.

Several major functions are described in the following subsections; in addition to them this unit exposes the functions, listed below. For Mac OS X and iOS:

- ToNSSTR(String):NSString – convert from Delphi string to NSString (added for consistency across Delphi versions).
- FromNSSTR(NSString):String and FromNSSTR(Pointer):String – convert from NSString to Delphi string; the NSString might be wrapped or unwrapped.
- PointerNSSTR(String):Pointer – same as NSSTR, included as part of the Foundation interface, but returning unwrapped NSString (especially useful for Apple collections).
- PointerNSObject(NSObject):Pointer – unwraps the object.

The following functions are only available for iOS:

- GetRootViewController:UIViewController – returns the current root view controller.
- SharedUIApplication:UIApplication – returns the shared UIApplication instance.

21.1 Device Unique Identifier

The following function returns an array of identifiers, which are unique for the current machine. On Mac OS X and Windows these are the MAC addresses of all the network adapters. On iOS 6 this is the identifierForVendor; on earlier iOS devices this is a persistent random number (meaning on iOS this value change on reinstall).

- GetDeviceUniqueIds:TArray<String> – usually the first array entry is good enough for most uses.

21.2 Implementation Existence Check

This helper allows you to verify whether the platform (Mac OS X or iOS) implements the given class. You can use this, for example, to check whether an iOS 6-only class is present.

- Defined – you can use this function by simply calling it on a given class (i.e. "if TSLComposeViewController.Defined then").

21.3 Apple Dictionary to String List Converter

This helper function converts a Mac OS X and iOS NSDictionary to Delphi's TStrings object:

- NSDictionaryToStrings(NSDictionary):TStrings

The general layout of the resulting TStrings is constructed to be easily accessible using the Values property. Example output might be:

```
Name1=Value1
Name2=Value2
Arr#=2          ← An array will have a count record with '#' suffix, followed by data
Arr[0].Name1=Value3
Arr[0].Name2=Value4
Arr[1]=Value5
```

Only NSDictionary and NSArray have specific converters, other types within use simple description converter.

21.4 Detaching Threads

This helper function works on all platforms and runs the given procedure in a separate thread, automatically managing the lifecycle of the underlying TThread object. On reference-counted platforms this function will host the list of active thread for correct operation.

- procedure DetachThread(TProc<TProc<TThreadProcedure>>);

To use the function, call it as follows:

```
DetachThread(procedure(Synchronize: TProc<TThreadProcedure>)
  begin
    Your_Code_Here;

    Synchronize(procedure
      begin
        Your_Synchronized_Code_Here;
      end);

    More_of_Your_Code_Here;

  end);
```

21.5 Idle Execution

This helper function executes the given procedure in the main thread, but at the next idle time. This procedure returns immediately, without executing the code, but queuing it for idle execution.

- procedure ExecuteInIdle(TProc);

21.6 Showing Messages

The following group of procedures provides message boxes, whose invocation does not block the calling thread (on iOS, without an additional message loop). The user choice (if applicable) is provided via a callback, instead of a function result.

In all the functions below, if Title is not provided, a default for the dialog type will be used instead.

The following function presents a regular dialog with an OK button only. It is convenient for simply displaying a message without blocking the app.

- `procedure ShowDialog(Msg[; DlgType[; Title]]);`

The following function presents a confirmation dialog with OK and Cancel buttons. It calls the Result callback with a Boolean, indicating whether OK was clicked.

- `procedure ShowConfirmation(Msg; [Title;] Result: TProc<Boolean>);`

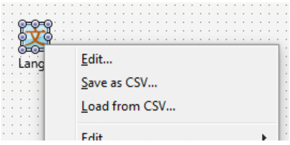
The following function is a generic one, allowing opening an arbitrary dialog, similarly to the standard Delphi function, but remaining non-blocking.

- `procedure ShowDialog(Msg; DlgType; [Title: string;] Buttons; [DefaultButton: TMsgDlgBtn;] Result: TProc<TModalResult>);`

21.7 Lang Saver

As an added convenience, for the developers creating multilingual applications, this component suite provides a design-time helper for the `TLang` Delphi component.

This helper is invoked by right clicking on the component icon in the designer; the following tools are added:



- Save as CSV...
- Load from CSV...

These tools allow you to save the content of the entire `TLang` component into a CSV file or load it from the CSV file. The file is generated for all languages, stored within the `TLang` component: each string is output as a line (i.e. Excel row) and each language is created as an additional column, across all values.

22. Content and Editions

This suite is supplied in four separate editions. This document refers to the Full edition, which includes all features and components. Other editions are described below.

22.1 Package Content

The package contains the following files and folders (the files below were created solely by JVEsoft ltd.):

- `Manual.pdf` – this technical manual.
- `JVE.*.pas` – the suite source code (see above for content).
- `JVEsoft.*` – runtime package files.
- `dclJVEsoft.*` – design time package files.

Several libraries are included with this distribution. All of them are freely available as downloads from their respected owners. The latest versions at the time of packaging were used. These files all have an extension "`a`" or "`jar`" and their names are self-explanatory. They are only used for iOS and Android platforms.

In addition to these, the following folders are included:

- `Component Icons` – self-explanatory.
- `Compiled Packages`, `Android`, `iOSDevice`, `iOSSimulator`, `OSX`, `Win32`, `Win64` – compiled component suite (due to download size considerations, only built using Delphi 10.4 Sydney; you can recompile them using another version).
- `Demo App` – a simple demo app, presenting some of the components at work.
- `Legacy` – some packages include a code version, compatible with older Delphi versions.

22.2 JVE Suite Editions

Four editions of the JVE Suite are available. They differ in the set of components, they provide. Separate manual or demo is not made for each of the editions, except for the Full one and the user is advices to purchase the Full edition.

In addition to the Full suite, the following more limited variations are included:

Analytics	**Ads Support**	**Native Tools**
4. Opening URLs 10. Google Analytics 18. Persistent Storage	4. Opening URLs 11. Chartboost, Facebook and AdMob 12. Banners and Interstitials 13. AdMob Support and Mediation 14. PubNative and MobFox 15. Native Ads Presentation	3. Embedded Browser 4. Opening URLs 5. Sending Emails 6. Social Networks Support 8. App Store Presentation 9. App Rating Reminder 18. Persistent Storage 19. In-Progress App Shading 20. Layout Controls

All editions are provided with their lifetime updates included.

23. License Agreement

JVEsoft Cross Platform Infrastructure Component Suite

You should carefully read the following terms and conditions before using this software. Your use of this software indicates your acceptance of this license agreement and warranty.

This software and its documentation are protected by the applicable copyright laws and by International Treaty provisions. Any use of this software in violation of copyright law or the terms of this agreement will be prosecuted to the best of our ability.

You are granted to make archival copies of JVEsoft Cross Platform Infrastructure Component Suite (the "Software") for the sole purpose of back-up and protecting your investment from loss. Under no circumstances may you copy this Software for the purposes of distribution to others. Under no conditions may you remove the copyright notices made part of the Software or documentation.

The supplied Software may be used by one person on as many computer systems as that person uses. Group programming projects making use of this Software must purchase a copy of the Software for each member of the group. Contact JVEsoft ltd. for volume discounts and site licensing agreements.

Software is licensed, not sold, to you! You may access the registered version of Software through a network, provided that you have obtained individual licenses for the Software covering all individuals that will access the software through the network. For instance, if 5 individuals will access Software on the network, each individual must have his own Software license, regardless of whether they use Software at different times or concurrently.

You may distribute, without run-time fees or further licenses, your own compiled programs based on any of the source code of the Software. You may not distribute any of the Software source code, compiled units, or compiled example programs without written permission from JVEsoft ltd. You may not use the Software to create components or controls to be used by other developers without written approval from JVEsoft ltd.

The Technical Manual is specifically excluded from the limitation above: you may redistribute the Technical Manual freely, by any means, as long as no modifications are made to it, including but not limited to the copyright notices therein.

Note that the previous restrictions do not prohibit you from distributing your own source code or units that depend upon the Software. However, others who receive your source code or units need to purchase their own copies of the Software in order to compile the source code or to write programs that use your units.

The Software is copyrighted and owned by JVEsoft ltd. You acknowledge that the title, ownership rights, and intellectual property rights in and to the Software shall remain in exclusive property of JVEsoft ltd.

JVESOFT LTD DOES NOT ASSUME ANY LIABILITY FOR THE USE OF THE JVESOFT CROSS PLATFORM INFRASTRUCTURE COMPONENT SUITE BEYOND THE ORIGINAL PURCHASE PRICE OF THE SOFTWARE. IN NO EVENT WILL JVESOFT LTD BE LIABLE TO YOU FOR ADDITIONAL DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE THESE PROGRAMS, EVEN IF JVESOFT LTD HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.