



William Duarte

# Delphi para Android e iOS

Desenvolvendo Aplicativos Móveis



**Prefácios de**  
Marco Cantù  
e Claudio Nasajon

**embarcadero**





**Delphi**  
**para**  
**Android e iOS**  
**Desenvolvendo Aplicativos Móveis**

Copyright© 2015 por Brasport Livros e Multimídia Ltda.

Todos os direitos reservados. Nenhuma parte deste livro poderá ser reproduzida, sob qualquer meio, especialmente em fotocópia (xerox), sem a permissão, por escrito, da Editora.

Para uma melhor visualização deste e-book sugerimos que mantenha seu software constantemente atualizado.

Editor: Sergio Martins de Oliveira

Diretora Editorial: Rosa Maria Oliveira de Queiroz

Gerente de Produção Editorial: Marina dos Anjos Martins de Oliveira

Editoração Eletrônica: SBNigri Artes e Textos Ltda.

Capa: Use Design

Produção de e-pub: SBNigri Artes e Textos Ltda.

Técnica e muita atenção foram empregadas na produção deste livro. Porém, erros de digitação e/ou impressão podem ocorrer. Qualquer dúvida, inclusive de conceito, solicitamos enviar mensagem para [brasport@brasport.com.br](mailto:brasport@brasport.com.br), para que nossa equipe, juntamente com o autor, possa esclarecer. A Brasport e o(s) autor(es) não assumem qualquer responsabilidade por eventuais danos ou perdas a pessoas ou bens, originados do uso deste livro.

Os nomes Delphi®, C++ Builder®, RAD Studio™ e Embarcadero são marcas comerciais ou marcas comerciais registradas da Embarcadero Technologies, Inc. Todos os seus direitos reservados.

ISBN Digital: 978-85-7452-755-0

**BRASPORT Livros e Multimídia Ltda.**

Rua Pardal Mallet, 23 – Tijuca

20270-280 Rio de Janeiro-RJ

Tels. Fax: (21) 2568.1415/2568.1507

**e-mails:**

[marketing@brasport.com.br](mailto:marketing@brasport.com.br)

[vendas@brasport.com.br](mailto:vendas@brasport.com.br)

[editorial@brasport.com.br](mailto:editorial@brasport.com.br)

---

site: [www.brasport.com.br](http://www.brasport.com.br)

**Filial**

Av. Paulista, 807 – conj. 915  
01311-100 – São Paulo-SP

William Duarte

# **Delphi** **para** **Android e iOS** **Desenvolvendo Aplicativos Móveis**



# Dedicatória

---

Este livro é dedicado primeiramente a Deus, que em sua infinita bondade e amor me capacitou para escrevê-lo.

Dedico a minha esposa, Priscila, que tem estado ao meu lado durante sete anos e sempre me estimulou a crescer. A você, meu amor, dedico este livro!

Dedico também ao meu amado pai, Jorge Duarte (in memoriam)

# Agradecimentos

---

A minha família, que tem dado o suporte e força necessários.

A todos os amigos da Nasajon Sistemas, do grupo Delphi<sup>®</sup> Experts no Facebook e a toda bela e crescente comunidade de desenvolvedores Delphi, mundo afora. Em especial ao Claudio Nasajon, que prontamente aceitou meu convite para escrever o prefácio deste livro.

Agradeço também a todos da Embarcadero Technologies, que ajudaram, e muito, na escrita deste livro. Um forte abraço para Fernando, Jim, Lisa e David, especialmente ao Marco Cantù, o maior guru Delphi<sup>®</sup> de todos os tempos, que gentilmente também aceitou o convite para escrever o prefácio deste livro.



# Apresentação

---

Este livro, amigo leitor, foi escrito com muito carinho e respeito a você. Nele, você encontrará um guia no melhor estilo passo a passo para desenvolver seus aplicativos móveis utilizando uma das maiores linguagens de programação já criadas, o Delphi®.

Para melhor aproveitamento deste livro, recomenda-se que você já tenha tido ao menos uma experiência com versões anteriores do Delphi®, uma vez que não serão abordados conceitos primários de programação, como laços, condicionais, variáveis, etc.

Fique à vontade para enviar suas dúvidas, críticas, elogios e ou reclamações (sim, reclamações também) para o meu e-mail:

[contato@williamduarte.com.br](mailto:contato@williamduarte.com.br).

# Sobre o Autor

---

William Duarte é *Delphi<sup>®</sup> Certified Developer* e *Embarcadero MVP*.

Iniciou sua carreira em 2005, como programador na empresa Pro Varejo, no Rio de Janeiro, sendo eleito funcionário do ano em 2006. Certificado internacionalmente como *Retail Technical Consultant* pela Retail Pro em 2007.

Especialista em automação comercial, atuou como colaborador nas principais empresas do país, como Quadrant Informática, sendo na época a maior empresa de software para o varejo da América do Sul, e na Linx Sistemas e Consultoria, uma das maiores empresas de software para o varejo do mundo. Em 2009, iniciou sua carreira na Nasajon Sistemas, eleita uma das melhores empresas para se trabalhar, também como desenvolvedor e também com tecnologias voltadas para o mercado varejista, sendo eleito o profissional do ano em 2011.

Atuou como instrutor de Delphi<sup>®</sup> e PAF-ECF na Digidata, prestando consultoria em diversos locais do Brasil, tendo treinado pelo menos cem alunos.

Em 2011 e 2014, participou como palestrante da Embarcadero Conference, em São Paulo. Nessa mesma época tornou-se colunista da revista Active Delphi, tendo escrito diversos artigos sobre varejo e Delphi<sup>®</sup>. No fim de 2012, a convite da Oracle of America, participou como consultor de varejo e auxiliou no processo de localização do software ponto de venda. Nesse mesmo ano, o software do qual sua equipe desenvolveu na Nasajon foi eleito “a melhor solução para o varejo” pela Associação das Empresas Brasileiras de Tecnologia da Informação (ASSESPRO).

Atualmente especializou-se em tecnologias móveis para o varejo, tendo participado, como palestrante, do 1º Congresso Virtual de TI, o CONVITI, com a palestra “Android para Força de Vendas”.

Mantém um web site onde aborda tecnologias móveis e desenvolvimento Delphi<sup>®</sup> que pode ser acessado através do link:

---

<http://www.williamduarte.com.br>.

# Prefácio por Cláudio Nasajon

---

Quando o conteúdo é excepcional, a forma perde importância, mas quando o conteúdo e a forma são excepcionais, você está diante de uma obra-prima.

Ao planejar uma obra, seja ela um livro ou um programa de computador, independentemente da plataforma, a sua primeira preocupação deve ser com o conteúdo. A estrutura geral e a lógica de programação têm impacto significativo no desempenho do produto final. Mas há detalhes de forma, como o número de linhas de código e até o tipo de variáveis que serão utilizadas, que podem fazer a diferença entre um resultado apenas medíocre e algo sensacional.

Você pode criar um *flag* usando uma variável numérica à qual atribui os valores “zero” e “um” conforme a situação, mas a boa prática de programação ensina que, nesses casos, o melhor é usar um *flag* “lógico” que assuma os valores “verdadeiro” e “falso”. O resultado final é o mesmo (ligado/desligado), mas consome muito menos espaço e roda muito mais rápido.

A estrutura dos *loops*, os tipos de variáveis e a estrutura geral do programa, fazem toda a diferença no funcionamento da aplicação.

Clareza e objetividade são importantes em qualquer trabalho. Quanto mais limpo o código, mais fácil é entendê-lo e alterá-lo. E sim, mesmo depois de pronto, é muito provável que você tenha que atualizar o código de tempos em tempos. Alguns com mais frequência do que outros, mas é uma lei da natureza: o mundo evolui e vez por outra precisamos ajustar o código para acompanhá-lo.

Uma aplicação feita com variáveis do tipo `ap01` e `ap02` ou `var01` e `var02b` vai levar muito mais tempo para ser interpretada por outros, ou mesmo por você, do que outra cujas variáveis sejam mais explicativas tais como `nomeCliente` ou `totalCompras`, por exemplo. Linhas de comentário ajudam bastante nesse terreno. Use-as sem moderação.

E existem práticas que atuam nas duas frentes, tanto na de melhoria do conteúdo quanto da forma. Por exemplo: se você tem uma rotina que se repete ao longo da aplicação, ao destacá-la (transformando-a em sub-rotina, chamada nos diversos pontos) você consegue não só simplificar o código (melhorando a forma) como eliminar pontos críticos de geração de problemas (melhorando o conteúdo). Quantas e quantas vezes não temos rotinas “repetidas” no programa e, ao atualizá-las, esquecemos alguns dos clones?

Comecei a minha carreira como programador no então Centro de Processamento de Dados do Estado do Rio de Janeiro, hoje PRODERJ, nos inícios dos anos 80. Certa vez fui chamado para a sala do meu chefe. Isso não era comum e fiquei um pouco tenso, mas fui desarmado. Chegando lá ele me esperava com a listagem do código-fonte do meu último projeto e perguntou-me a razão de eu ter usado variáveis em latim (ou o que eu interpretava como sendo latim). Sim, eu havia usado variáveis como “totvs-totatis” e “nomen-ipsium” (nome do cliente) por exemplo. Eu não tinha muita explicação. No auge dos meus 19 anos eu só achava aquilo divertido. Ele então me deu uma lição que jamais esqueci: o código que você desenvolve em um programa de computador é uma espécie de legado. Ele pode durar décadas e é possível que chegue o tempo em que você não estará mais por perto quando precisar fazer ajustes. Então, ao programar uma aplicação, pense primeiro em você, pois daqui a alguns anos não vai mais se lembrar da estrutura como lembra hoje e variáveis sem sentido não farão sentido algum, mas também pense no seu legado, nos outros que lhe seguirão para manter o animal vivo e em condições. Você pode ser o pai ou a mãe da criança, mas outros provavelmente serão os seus tutores. Crie o seu “filho” para o mundo, não apenas para você.

Quando o William me pediu para escrever este prefácio eu me senti muito honrado e prontamente aceitei sem sequer conhecer o conteúdo do livro. Vindo dele, só poderia ser coisa boa. Quando o texto ficou pronto, pedi a ele para ler antes de escrever qualquer coisa. Queria saber o que, exatamente, eu estaria prefaciando. O resultado não poderia ter sido melhor, o que só confirmou a minha suspeita inicial de que o trabalho seria uma “obra-prima”, com conteúdo e forma excepcionais.

William Duarte foi um dos melhores desenvolvedores com quem tive o privilégio de trabalhar. Competente não só na forma de programar, mas também na

atitude, sempre querendo entender o contexto onde o desenvolvimento seria aplicado. Conteí com ele em diversas ocasiões como “consultor” para ajudar a elaborar a especificação funcional de sistemas complexos e sempre o vi como uma pessoa com quem podemos contar em nossa organização.

Ao escrever essas palavras para apresentar a sua obra, eu faço um reconhecimento público da sua competência como desenvolvedor e da sua habilidade como comunicador. Torço para que você, ao ler as linhas que seguem a este prefácio, entenda o porquê dessa admiração e, principalmente, espero que você possa aplicar esse conhecimento para criar o seu próprio legado.

Boa leitura!

**Claudio Nasajon**, Nasajon Sistemas

# Prefácio por Marco Cantù

---

Over that last 20 years, Delphi has been one of the leading development tools for building applications for the Microsoft Windows operating system and it now offers a unique opportunity for mobile development. Initially produced by Borland, and now owned by Embarcadero Technologies, the Integrated Development Environment has gone from a Windows-focused tool to a multi-device cross-platform and mobile solution.

The Windows side of the product is powered by two compilers (for 32-bit and 64-bit applications) and the best Windows components library, the Visual Components Library (VCL). As many of you know, I wrote several books (very popular in Brazil) covering Delphi and the VCL. But that was quite some time ago.

Embarcadero has been investing a lot in new development for Delphi turning it into a fantastic mobile tool. The mobile side of the product is currently powered by four compilers (Android, iOS 32-bit, iOS 64-bit, and iOS simulator) and a state-of-the-art and unique visual and cross-platform library, called FireMonkey. Combined the multi-device solution is based 7 compilers (there is also one for OS X) and a library of components and visual controls that encompasses the four supported operating systems.

Why this change in focus and why does this matter to developers? While Windows is still a fundamental platform for the development of business applications, the Microsoft OS is currently considered the third most popular operating system, after two mobile ones, Android and iOS. With billions of users, the mobile environments have taken the lead in terms of distribution, and are now the preferred computing device not only for communication needs, entertainment, and games, but are becoming more and more critical for business software development, alone or combined with Windows solutions. While Android leads in terms of

pure numbers, iOS attracts many high-end users, and so they both offer great revenues opportunities.

With this change in the IT world, Delphi also needed to change and it did very timely. Based on the solid foundations of a very modern language (far extended and expanded from what it was years), a large set of compilers, a solid multi-platform runtime library, Delphi is the tool that can get you into the mobile world, without giving up on rapid and effective RAD development and a component-based system.

Delphi offers a great and fast transition for existing developers to embrace the mobile revolution, providing a familiar environment and many shared components. It also offers a great traction for mobile developers looking for a tool that offers real multi device support without sacrificing the native integration.

Still, moving to mobile for Delphi developers or moving to Delphi for mobile developers takes some effort and a good understanding of the mobile world and of the foundations of the FireMonkey library. This is a reason why it is great to see books coming out with the specific goal of teaching developers to work with Delphi for building mobile apps. William Duarte's book on "Developing mobile applications in Delphi for Android and iOS" is a book you need to get the proper understanding of this new development world. I met him in São Paulo a few months ago, he explained me about this book, and I really like his approach in teaching how to best use the product, not touching only on the surface and the easy way of building apps, but going behind the scenes and including a lot of technical details to help you fully understand what you are doing and why.

The coverage of Delphi is extensive, but the book also encompassed all you need for a great development experience. For example, you'll start by learning how to configure your IDE and the required SDK tools for the mobile platforms, how to deploy to your mobile devices and the app stores. The days of a single platform were certainly easier for developers, compared to today's landscape. Of course the core of the book covers actual development with FireMonkey, building user interfaces, using Live Bindings to show data, managing lists of items with the ListView component, creating responsive and flexible UIs, accessing to databases, and many other standard elements of applications. While many topic might seem



familiar, there are many specific of the mobile world you'll learn about, and many tips that will speed up your development efforts.

Phones and tablets have also a much rich interaction with the world around them, compared to regular PCs. From using cameras to sensors, from GPS and localization to device movement, from device notifications to making actual phone calls, there is a lot to learn and a lot in the book providing you with the information you need to get up to speed with so many different technologies. Of course, this new world of apps is also a world in which monetization often doesn't come from licensing, but from advertising, another area covered in the book.

While the single-source cross-platform development model is at the core of Delphi (so you avoid writing multiple different application for each of the mobile platforms), the uniqueness of the solution is that it also allows hooking to the Java and Objective-C native libraries provided by Android and iOS. This is obviously not a simple topic, and it is great that the book devotes a significant space to it. One of my favorite features of Delphi is AppTethering, which is the ability to easily let an existing VCL application interact with one running on a phone or tablet, in a simple and smooth way, has also its own chapter.

As a developer you cannot ignore mobile, and Delphi is a great tool for mobile development today. Fire up the IDE, start reading this book, and enjoy the new world of mobile. The App Store and Play Store are waiting for your apps, and I'm looking forward to hear about even more great apps powered by Delphi. The mobile world is yours to conquer once you have the right tool, Delphi, and the right instructor, William Duarte, helping you!

### **Marco Cantù**

RAD Studio Product Manager at Embarcadero Technologies

Author of the Mastering Delphi series and of the Object Pascal Handbook

Piacenza, Italy

marco.cantu@gmail.com

### **TRADUÇÃO:**

Durante os últimos 20 anos, o Delphi tem sido uma das principais ferramentas de desenvolvimento para criar aplicativos para o sistema operacional Microsoft Windows e agora oferece uma oportunidade única para o desenvolvimento

*mobile*. Inicialmente produzido pela Borland, hoje propriedade da Embarcadero Technologies, o ambiente de desenvolvimento integrado (IDE) do Delphi passou de uma ferramenta focada em Windows para uma ferramenta multiplataforma e para dispositivos móveis.

No Windows, o produto detém dois compiladores (para aplicativos de 32-bit e 64-bit) e a melhor biblioteca de componentes, a *Visual Components Library* (VCL). Como muitos de vocês já sabem, eu escrevi vários livros (muitos populares no Brasil) que cobriam o Delphi e o VCL. Mas isso já faz tempo.

A Embarcadero tem investido muito em um novo desenvolvimento para o Delphi, transformando-o em uma ferramenta para *mobile* fantástica, atualmente composta por quatro compiladores (Android, iOS 32-bit, iOS 64-bit e simulador iOS) e uma biblioteca visual multiplataforma com tecnologia de ponta chamada FireMonkey. Combinando a solução multidispositivo, são sete compiladores (há também um para OS X) e uma biblioteca de componentes e controles visuais que englobam os quatro sistemas operacionais suportados.

Por que essa mudança de foco e por que isso importa para os desenvolvedores? Embora o Windows ainda seja uma plataforma fundamental para o desenvolvimento de aplicativos de negócios, atualmente ele é considerado o terceiro sistema operacional mais popular, depois do Android e iOS. Com bilhões de usuários, os ambientes móveis assumiram a liderança em termos de distribuição e agora são os dispositivos de computação preferidos não só para as necessidades de comunicação, entretenimento e jogos, mas estão se tornando cada vez mais críticos para o desenvolvimento de software de negócios, combinados ou não com soluções Windows. Apesar de o Android liderar em termos numéricos, o iOS atrai muitos usuários *high-end*, e assim ambos oferecem grandes oportunidades de receita.

Com esta mudança no mundo da TI, o Delphi também precisava mudar, e assim o fez em um momento muito oportuno. Com base nos fundamentos sólidos de uma linguagem moderna (agora estendida e expandida), um grande conjunto de compiladores, uma biblioteca de tempo de execução multiplataforma sólida, o Delphi é a ferramenta que você quer para o mundo *mobile*, sem desistir do desenvolvimento RAD, rápido e eficaz, e de um sistema baseado em componentes.

O Delphi oferece uma grande e rápida transição para desenvolvedores abraçar em a revolução do *mobile*, proporcionando um ambiente familiar e muitos

componentes compartilhados. Ele também oferece uma excelente interação para os desenvolvedores *mobile* que procuram uma ferramenta que ofereça verdadeiro suporte multidispositivo sem sacrificar a integração nativa.

Ainda assim, migrar para o mundo *mobile* sendo desenvolvedor Delphi, ou migrar para o Delphi sendo desenvolvedor *mobile*, requer algum esforço e um bom entendimento do mundo *mobile* e dos fundamentos da biblioteca FireMonkey. Esta é uma razão pela qual é ótimo encontrar publicações com o objetivo específico de ensinar os desenvolvedores a trabalhar com Delphi para a construção de aplicativos móveis. O livro do William Duarte traz o que você precisa para obter a compreensão adequada desse novo mundo. Eu o conheci em São Paulo há alguns meses e quando ele me explicou sobre este livro eu gostei muito de sua abordagem sobre como usar melhor o produto, não tocando apenas superficialmente e da forma mais fácil na construção de aplicações, mas indo por trás dos panos e incluindo uma série de detalhes técnicos para lhe ajudar a entender completamente o que você está fazendo e por quê.

A cobertura do Delphi é extensa, mas o livro engloba tudo o que você precisa para uma grande experiência de desenvolvimento. Por exemplo, você vai começar aprendendo a configurar o IDE e as ferramentas do SDK necessárias para as plataformas móveis, como instalar em seus dispositivos móveis e nas lojas de aplicativos. Comparado ao panorama atual, os dias de uma única plataforma eram certamente mais fáceis para os desenvolvedores. O núcleo do livro abrange o desenvolvimento real com FireMonkey, a construção de interfaces com o usuário usando *Live Bindings* para mostrar dados, gerenciamento de listas de itens com o componente *ListView*, criando responsivas e flexíveis interfaces de usuário, acesso a bases de dados e muitos outros elementos padrões dos aplicativos. Enquanto vários tópicos podem parecer familiares, há muitos específicos do mundo móvel sobre os quais você vai aprender e muitas dicas que irão acelerar seus esforços de desenvolvimento.

Telefones e *tablets* também têm uma interação muito rica com o mundo ao seu redor, em comparação aos PCs comuns. Desde o uso de câmeras a sensores, de GPS e geolocalização para o movimento do dispositivo, de notificações para chamadas telefônicas reais, há muito para estudar e este livro irá lhe proporcionar as informações que você necessita para aprender depressa com tantas tecnologias diferentes. Claro, este novo mundo de aplicativos também é um mundo no qual a

monetização muitas vezes não vem de licenciamento, mas de formar publicidade, outra área coberta no livro.

Enquanto o modelo de desenvolvimento multiplataforma em um único código-fonte é o cerne do Delphi (para que você evite escrever diferentes aplicativos para cada uma das plataformas móveis), a singularidade da solução é que ele também permite interagir com bibliotecas nativas em Java e Objective-C fornecidas pelo Android e iOS. Isso obviamente não é um tema simples, e é ótimo que este livro dedique um espaço significativo para o assunto. Um dos meus recursos favoritos do Delphi é o *AppTethering*, que tem a capacidade de deixar facilmente uma aplicação VCL existente interagir com uma aplicação em um celular ou *tablet*, de forma simples e suave, e também possui seu próprio capítulo.

Como desenvolvedor você não pode ignorar o *mobile*, e hoje o Delphi é uma grande ferramenta para esse tipo de desenvolvimento. Abra seu IDE, comece a ler este livro e desfrute do novo mundo *mobile*. As App Store e Play Store estão à espera de seus aplicativos, e eu estou ansioso para ouvir ainda mais sobre excelentes aplicativos desenvolvidos em Delphi. O mundo *mobile* é seu para conquistá-lo. Você tem a ferramenta certa, Delphi, e o instrutor certo, William Duarte, para ajudá-lo!

**Marco Cantù**

RAD Studio Product Manager at Embarcadero Technologies

Autor da série Mastering Delphi e do Object Pascal Handbook

# Sumário

---

## **1. Conceitos Básicos do FireMonkey**

### **Primeiros passos**

#### **Requisitos e pré-requisitos para o Delphi® XE7 e XE8**

[Plataformas Windows](#)

[Pré-requisitos](#)

[Para o desenvolvimento de aplicativos de 64-bits do Windows](#)

[Para o desenvolvimento de aplicações OS X](#)

[Para o desenvolvimento de aplicações iOS](#)

[Para o desenvolvimento de aplicações Android](#)

[Versões do Android suportadas](#)

## **2. Configurando o Ambiente para Desenvolvimento iOS**

### **Instalar Xcode no Mac OS**

### **Instalar o Platform Assistant**

### **Executar o assistente de plataforma**

### **Instalar o Xcode Command Line Tools**

### **Pedido, download e instalação do certificado de desenvolvimento**

### **Registre o seu dispositivo para deploy**

### **Criar e instalar um perfil de provisionamento**

### **Configurar o Delphi® para compilar em um dispositivo ou simulador iOS**

### 3. Configurando o Ambiente para Desenvolvimento Android

Instalação do SDK e NDK do Android no RAD Studio

Instalar o controlador USB para o seu dispositivo Android

Ativando o modo de depuração USB no dispositivo

Criando um novo emulador

Criando uma nova conta na Google Play

### 4. Hello World

### 5. Componentes Básicos para Interface Visual

Uso de componentes nativos

Botões com diferentes estilos

Layouts responsivos – Diferentes telas, diferentes dispositivos

Criando layouts com Scroll

Scroll layout: TScrollBox

Utilizando botões em barra de ferramentas

Utilizando texto com teclados diferentes

Criando menus com TMultiView

Browser – Componente para navegação web

### 6. ListView e LiveBindings

Iniciando o projeto

Adicionando campos

Manipulando o LiveBindings

Adicionando mais campos

Adicionando o evento onClick

Resultado

## 7. ListBox

Lista plana e lista agrupada

Adicionar um item ao ListBox

## 8. Acessando a Câmera

TTakePhotoFromCameraAction, a forma mais fácil

TCameraComponent – Princípios básicos

TCameraComponent – Utilizando o flash

TCameraComponent – Câmera frontal e traseira

## 9. Sensores

TCustomSensor – Passos iniciais

Sensor de luminosidade

Sensor magnético

Sensor de movimento

Sensor de localização

Geocoding e reverse geocoding

## 10. Notificações

Acesso ao serviço de notificações

Apresentar uma notificação agendada

Atualizar ou cancelar uma notificação agendada

## 11. Permissões e Opções do Projeto

Permissões padrão

AndroidManifest.xml

Opções do projeto

## 12. API de Chamadas

[Primeiros passos: informações da operadora](#)

[Realizando uma chamada](#)

[Detectando o estado da chamada](#)

### [13. FireDAC e SQLite](#)

[Conceitos básicos](#)

[Conexão com o FireDAC](#)

[BeforeConnect e AfterConnect](#)

[FireDAC com LiveBindings](#)

[Configurar seu banco de dados no dispositivo móvel](#)

[Executando seu aplicativo](#)

### [14. Utilização de Linguagem Nativa](#)

[Usando classes Java do SDK Android](#)

[Exemplo – Atalho do aplicativo na área de trabalho](#)

[Toast Message – Android](#)

[Acessando wi-fi via API nativa no Android](#)

[Usando classes Objective-C no iOS](#)

[Acessando wi-fi via API nativa no iOS](#)

### [15. App Tethering](#)

[Primeiros passos – Aplicativo servidor](#)

[A mágica – Aplicativo cliente](#)

### [16. Media Player](#)

[Aperte o play](#)

[Aperte o rec](#)

### [17. Arquivos INI](#)



[TMemIniFile](#)

## [18. Monetizar sua Aplicação](#)

[AdMob](#)

[iAd](#)

[TBannerAD](#)

## [19. Publicando seu Aplicativo](#)

[Deployment Manager – Preparando seu aplicativo para publicação](#)

[Recursos do Deployment Manager](#)

[Publicando seu aplicativo na App Store](#)

[Publicando seu aplicativo na Google Play](#)

## [Apêndice. Delphi XE8 – Visão Geral](#)

[O que há de novo?](#)

[Novo cliente HTTP](#)

[Beacons](#)

[Castalia](#)

## [Considerações Finais](#)

## [Referências Bibliográficas](#)

# 1. Conceitos Básicos do FireMonkey

---

## Primeiros passos

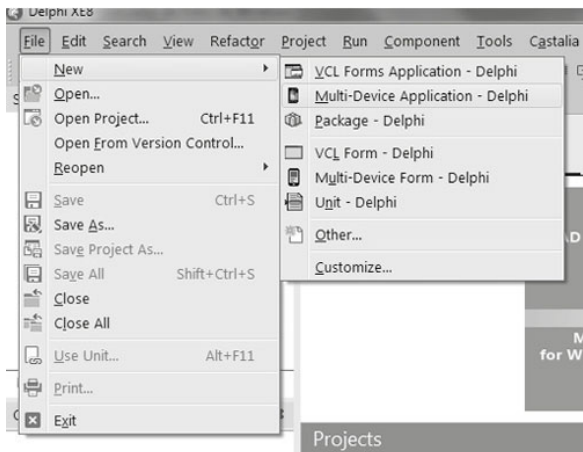
O FireMonkey (FMX) é o *framework* e plataforma de desenvolvimento do Delphi®, C++ Builder® e RAD Studio™, desde a versão XE2. O FMX permite que o mesmo código-fonte seja compilado para diversos dispositivos, gerando código de máquina nativo. Atualmente na versão XE8, o FireMonkey gera aplicativos nativos para Windows, Mac, Android e iOS.

A VCL (*Visual Component Library*) não foi descontinuada e não existem rumores de que será. Para criar um novo projeto, você pode selecionar entre trabalhar com a VCL ou com o FMX – lembrando, é claro, que somente o FMX permite o desenvolvimento multiplataforma, enquanto o VCL permite apenas para Windows.

Para criar um novo projeto FireMonkey, abra sua IDE (*Integrated Development Environment*) do Delphi® e selecione a opção *File > New > Multi-Device Application – Delphi*.

Em seguida, será solicitado que você escolha uma das opções de aplicação padrão, que são, nesta ordem:

- **Blank Application** – O modelo de aplicativo em branco é um ponto de partida para criar um aplicativo completamente customizável.
- **3D Application** – O modelo de aplicativo 3D é um ponto de partida para a criação de um aplicativo 3D multidispositivo baseado em um *Form 3D*.



**Figura 1 – IDE do Delphi®.**

- **Header/Footer** – Este modelo consiste em um cabeçalho/rodapé da barra de ferramentas e um *Toolbar*. Ele serve como um bom ponto de partida para a sua aplicação multidispositivos.
- **Header/Footer with Navigation** – Consiste em um cabeçalho/rodapé com uma barra de ferramentas. Contém um *TabControl* com duas abas para navegação. Você também pode adicionar mais abas para sua aplicação.
- **Master-Detail** – Contém um controle *MultiView*. A visão mestre exibe os dados na lista principal e a visão detalhe exibe as informações de detalhes para o item selecionado. Semelhante ao menu estilo gaveta (*Drawer*) do Facebook. O componente *ListView* fornece várias opções de estilo através da propriedade *ItemAppearance*.
- **Tabbed** – O modelo com abas consiste em um cabeçalho/*Toolbar* e quatro abas. Você também pode adicionar mais abas para a sua aplicação.

- **Tabbed with Navigation** – O modelo com abas navegáveis consiste em um cabeçalho/*Toolbar* e quatro abas. A primeira página contém um *TabControl* aninhado, que por sua vez contém duas guias filhas. Você também pode adicionar mais abas para a sua aplicação.

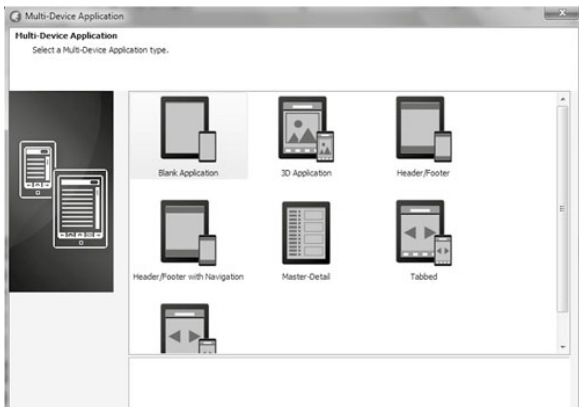


Figura 2 – Seleção de projeto multidispositivo.

## Requisitos e pré-requisitos para o Delphi® XE7 e XE8

### Plataformas Windows

O Delphi® originalmente foi desenvolvido para ser executado apenas em plataformas Microsoft® Windows. Portanto, para o ambiente de desenvolvimento, você deverá possuir um computador com o sistema operacional da Microsoft® instalado.

- Microsoft® Windows 8 ou 8.1 (32-bit e 64-bit).
- Microsoft Windows 7 com SP 1 (32-bit e 64-bit).
- Os últimos *service packs* e atualizações de segurança são recomendados para todas as plataformas.

- Microsoft Internet Explorer 6.0 SP 1 ou superior.

## Pré-requisitos

O Delphi exige vários pré-requisitos que são normalmente instalados automaticamente em seu sistema – em particular, o Microsoft .NET Framework, que provavelmente estará instalado em seu sistema operacional, se ele já não estiver presente (a menos que você esteja instalando no Windows 8).

Os pré-requisitos incluem:

- Microsoft .NET Framework 3.5.
- Microsoft Visual J# version 2.0 Redistributable Package (32-bit).
- Microsoft Document Explorer.

O .NET Framework deve ser instalado automaticamente. No entanto, se você estiver instalando no Windows 8, isso não ocorre. Você pode optar por instalar o .NET Framework via MS Windows Update Server.

## Para o desenvolvimento de aplicativos de 64-bits do Windows

Ou um PC executando uma versão de 64-bits do Windows, ou um PC para o desenvolvimento de 32-bits conectado a um PC executando uma versão de 64-bits do Windows.

## Para o desenvolvimento de aplicações OS X

Um PC com Windows ligado através de uma rede local (usando um SSH, VNC ou solução de compartilhamento de arquivos do Windows) em um Mac baseado em Intel com 2 GB de RAM ou mais, executando um dos seguintes sistemas operacionais:

- OS X 10.8 (Mountain Lion).
- OS X 10.9 (Mavericks).
- OS X 10.10 (Yosemite).

## Para o desenvolvimento de aplicações iOS

Além dos requisitos para aplicações OS X, um dispositivo iOS (iPhone, iPad ou iPod Touch) conectado ao cabo USB Mac pela Apple, com a versão apropriada do Xcode instalado:

- Xcode 5 para o iOS 7, ou Xcode 5.1 para o iOS 7.1.
- Xcode 6 para o iOS 8.

## Para o desenvolvimento de aplicações Android

Um PC com Windows conectado a um dispositivo Android suportado (ou seja, rodando em um processador ARMv7 com suporte NEON).

## Versões do Android suportadas

| Android OS         | Versão                      | API Level           |
|--------------------|-----------------------------|---------------------|
| KitKat             | Android 4.4                 | API Level 19        |
| Jelly Bean         | Android 4.1.x, 4.2.x, 4.3.x | API Levels 16,17,18 |
| Ice Cream Sandwich | Android 4.0.3 e 4.0.4       | API Level 15        |
| Gingerbread        | Android 2.3.3 – 2.3.7       | API Level 10        |

**Aviso: não são suportadas as seguintes configurações:**

**Honeycomb (Android 3.x).**

**Gingerbread emulator.**

**Tegra 2 Devices (NEON é requerido).**

CPUs ARM são criados por uma grande variedade de fabricantes, de acordo com diferentes conjuntos de especificações. ARM é a tecnologia dominante em hardware móvel. O conjunto de instruções ARMv7, ou núcleo, especifica a microarquitetura que a CPU usa.

A família de CPUs que implementar instruções ARMv7 é chamada de *Cortex-A Series*. A maioria desses CPUs também inclui tecnologia NEON. Alguns dispositivos Android usam a arquitetura Intel ATOM, o que é incompatível com o FireMonkey. A razão pela qual as aplicações do RAD Studio™ exigem essa arquitetura específica de CPU é que os aplicativos do RAD Studio compilam um código de máquina para melhor desempenho. Isso é diferente de outros

---

aplicativos do Android que são desenvolvidos com Java que compilam para o *Dalvik bytecode* e assim executam na máquina virtual Dalvik.

A maioria dos dispositivos Android possui uma GPU, ou *Graphics Processing Unit*. É comum a GPU ser combinada com a CPU, em uma configuração *System-on-a-Chip* (ou SoC).

## 2. Configurando o Ambiente para Desenvolvimento iOS

---

Para configurar seu ambiente de desenvolvimento a fim de desenvolver seus aplicativos suportando o iOS, é necessário ter uma instalação do sistema operacional Mac OS X em seu computador, conforme descrito nos requisitos.

O primeiro passo é criar sua conta de desenvolvedor na Apple, a *developer account*. Caso você já possua seu *Apple ID*, faça o *download* do Xcode através da App Store em seu Mac.

De acordo com a plataforma iOS selecionada, uma versão do Xcode diferente será necessária, por exemplo:

- Xcode 5 para o iOS 7, ou Xcode 5.1 para o iOS 7.1.
- Xcode 6 para iOS 8.

Existem duas formas de testar seus aplicativos para iOS: a primeira delas, e gratuita, é através do *Apple Developer Program*. Nessa modalidade só é possível testar seus aplicativos através do *iOS Simulator*.

Para maiores informações, entre no website da Apple:

<https://developer.apple.com/register/>.

O registro no *Apple Developer Program* tem as seguintes vantagens:

- Acesso às ferramentas de softwares básicos (como Xcode e iOS SDK) de que você precisa para desenvolver aplicativos para iOS.
- Permite testar seus aplicativos iOS usando simuladores no Xcode.
- Acesso à documentação da Apple sobre o desenvolvimento iOS.
- Permite que você relate *bugs* e participe de eventos de desenvolvedores da Apple.



A segunda forma, com um custo anual de 99 dólares americanos a serem pagos à Apple, chama-se *iOS Developer Program*. Nessa modalidade, além de testar seus aplicativos em simuladores, você poderá enviar seu aplicativo para a App Store e testar em seu dispositivo físico.

O registro no *iOS Developer Program* tem as seguintes vantagens:

- Acesso a ferramentas de desenvolvimento, software de pré-lançamento e suporte técnico.
- Permite que você coloque suas aplicações iOS concluídas na App Store.
- Acesso ao portal através do qual você pode criar o perfil de desenvolvimento *Provisioning*, que é necessário para a execução de seus aplicativos para dispositivos iOS.

Existem programas de desenvolvedores independentes para empresas (*business*) e para universidades. Para obter informações sobre como ingressar no *iOS Developer Program*, veja <https://developer.apple.com/support/enrollment/>.

## Instalar Xcode no Mac OS

Após criar sua conta de desenvolvedor na Apple, é necessário instalar o Xcode. Para isso basta ir na App Store, em seu Mac.



**Figura 3 – Download do Xcode.**

Xcode é o ambiente de desenvolvimento e depuração no Mac e fornece os arquivos de desenvolvimento necessários aos aplicativos do Mac OS X e iOS.

Você pode instalar Xcode a partir de qualquer das seguintes fontes:

- Em seu DVD *Mac OS X Install*, sob “Instalações Opcionais”, clique duas vezes em *Xcode.mpkg* para instalar o Xcode em seu sistema.
- Na App Store, baixar o Xcode gratuitamente.
- Como *Apple Developer* registrado, você pode baixar a última versão do Xcode como um *bundle* (.dmg).

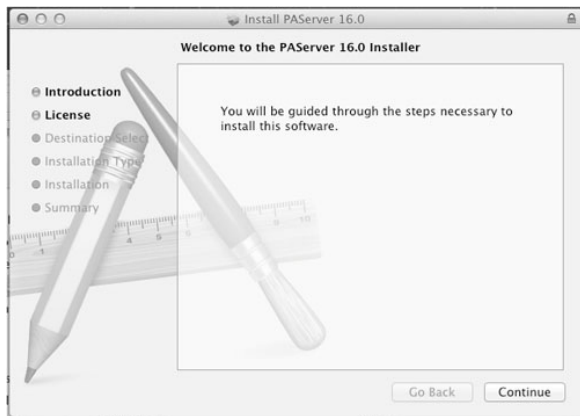
## Instalar o *Platform Assistant*

O *Platform Assistant* (assistente de plataforma) é uma espécie de ponte que conecta o compilador Delphi® a um outro computador com Mac OS X. A premissa é que seu ambiente de desenvolvimento integrado (IDE) esteja em um computador com Windows e seu ambiente de testes, executando um Mac OS X, onde será instalado o *Platform Assistant*.

O *Platform Assistant* deve estar rodando no Mac quando você faz um *deploy* de um App iOS a partir do seu PC, quer ao simulador iOS ou a um dispositivo físico iOS.

O instalador do Mac OS X para o *Platform Assistant* é nomeado *PAServer-15.0.pkg* e está disponível nos seguintes locais:

- C:\Program Files (x86)\Embarcadero\Studio\15.0\PAServer\PAServer15.0.pkg.
- <http://altd.embarcadero.com/release/studio/15.0/PAServer/PAServer15.0.pkg>.
- [http://altd.embarcadero.com/release/studio/15.0/PAServer/PAServer15.0\\_update1.pkg](http://altd.embarcadero.com/release/studio/15.0/PAServer/PAServer15.0_update1.pkg) (somente se estiver com o *update 1* do XE7).
- <http://altd.embarcadero.com/release/studio/16.0/PAServer/PAServer16.0.pkg> (XE8).
- [http://docwiki.embarcadero.com/RADStudio/XE8/en/Installing\\_the\\_Platform\\_Assistant\\_on\\_a\\_Mac](http://docwiki.embarcadero.com/RADStudio/XE8/en/Installing_the_Platform_Assistant_on_a_Mac) (links para *update*) ou na pasta de instalação do Delphi – por exemplo: C:\Program Files (x86) \Embarcadero\Studio\16.0\PAServer\.

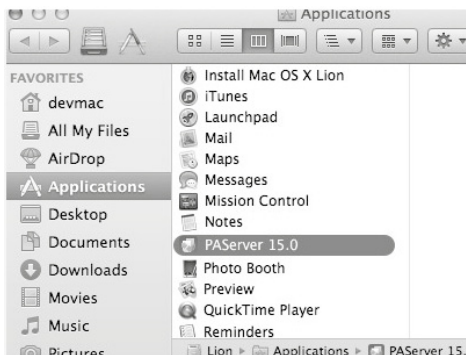


**Figura 4 – Instalação do *Platform Assistant*.**

## Executar o assistente de plataforma

No *Finder*, ativar o arquivo (.App) (*PA Server 15.0.app*) como segue:

1. Navegue até a pasta “Aplicativos” de nível superior.
2. Clique duas vezes em *PA Server 15.0.app* para iniciar o Assistente de Plataforma.
3. Em seguida, você será solicitado a digitar sua senha de usuário Mac para permitir que o Assistente de Plataforma realize o *debug* da sua aplicação.



**Figura 5 – Execute o PA Server 15.0.**

Quando solicitado, preencha o nome do usuário e a senha para que o *Platform Assistant* “assuma o controle”.



**Figura 6 – Dê permissão para o PAServer “assumir o controle”.**

Seguindo todos os procedimentos, será exibido terminal conectado na porta 64211, onde será feita a conexão do Delphi<sup>®</sup>, no Windows, para o Mac OS X com o Xcode realizar o *deploy*.

```
Platform Assistant Server Version 6.0.2.15
Copyright (c) 2009-2014 Embarcadero Technologies, Inc.

Connection Profile password <press Enter for no password>:

Acquiring permission to support debugging...succeeded

Starting Platform Assistant Server on port 64211

Type ? for available commands
>
```

**Figura 7 – PAServer em execução na porta 64211.**

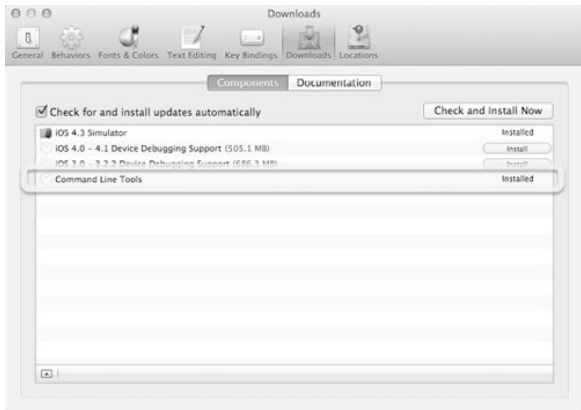
## Instalar o Xcode *Command Line Tools*

Para instalar um aplicativo em seu dispositivo iOS para fins de depuração e testes, o Xcode deve instalar as ferramentas *Command Line Tools*. Para executar aplicativos iOS em um dispositivo iOS, este deve ser conectado via cabo USB para o Mac.

**Nota: a partir do Xcode versão 6.1, as ferramentas de linha de comando são instaladas automaticamente.**

Para instalar o *Command Line Tools*, siga os passos:

1. Inicie o Xcode no Mac.
2. Escolha “Preferências” no menu do Xcode.
3. Na janela “Geral”, clique na guia “Locais”.
4. Verifique se as ferramentas de linha de comando mostram a versão do Xcode com a qual foram instaladas.
5. Se as ferramentas de linha de comando Xcode não estiverem instaladas, a versão do Xcode não aparecerá na linha de comando.



**Figura 8 – Instalação do Xcode *Command Line Tools*.**

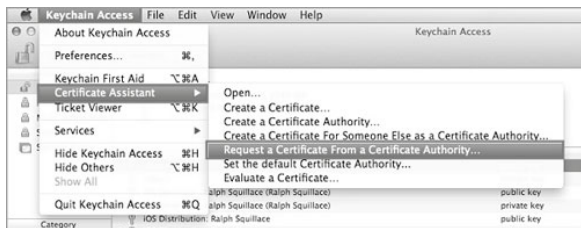
## Pedido, *download* e instalação do certificado de desenvolvimento

Aplicações que são instaladas no dispositivo (ou no *iOS Simulator*) precisam ser assinadas antes de executadas. O certificado de desenvolvimento contém informações que são necessárias para a assinatura das aplicações. Cada indivíduo (um desenvolvedor individual ou um membro da equipe) deve ter um certificado de desenvolvimento único, que pode ser usado para várias aplicações.

Os certificados de desenvolvimento devem ser solicitados por cada membro da equipe, e esses pedidos devem ser aprovados por um administrador de equipe.

**Nota: essa modalidade só está disponível através do registro no *iOS Developer Program*.**

Na aplicação *Keychain Access* no seu Mac, selecione, no menu *Keychain Access*, *Certificate Assistant* > *Request a Certificate From a Certificate Authority...*



**Figura 9 – Solicitando um certificado.**

Se você é um membro da equipe de desenvolvimento de um programa corporativo/organização, o administrador da equipe precisa aprovar o seu pedido. Depois que o administrador da equipe aprová-la, você pode baixar o certificado.

Se você é um desenvolvedor individual, você deve ver uma opção de *download* para o seu certificado logo depois que você pedir. Consulte a documentação da Apple em: <https://developer.apple.com/library/ios/documentation/Security/Conceptual/CodeSigningGuide/Introduction/Introduction.html>

Salve a solicitação de certificado como um arquivo CSR. Em seguida, envie-a para sua autoridade de certificação através do portal

<https://developer.apple.com/ios/my/certificates/index.action> da seguinte maneira:

1. Quando solicitado, digite o seu *Apple ID* e senha e clique em “Entrar”.
2. Sob “iOS Apps”, clique em “Certificados”.
3. Na página que se abre, clique no ícone de sinal de mais (+). Isso abre o Assistente para adicionar *iOS Certificate*.
4. Na página “Selecionar Tipo”, clique em “Continuar” e siga as instruções na tela para continuar com o assistente.
5. Quando solicitado, fazer o *upload* do arquivo CSR que você salvou no seu Mac.
6. Ir para *iOS Provisioning Portal*. Você pode baixar o certificado de desenvolvimento clicando no botão “Download”.
7. Executar o certificado de desenvolvimento clicando duas vezes nele. Ele carrega automaticamente no aplicativo *Keychain Access*.

## Registre o seu dispositivo para *deploy*

Antes de um dispositivo poder executar aplicativos do usuário, ele deve ser registrado no portal *Provisioning*, da Apple. Os dispositivos são registrados pelo seu *Unique ID* do dispositivo (UDID). O UDID pode ser determinado usando o Xcode.

1. Verifique se o seu dispositivo iOS está ligado à sua máquina Mac.
2. Abra o Xcode e vá para *Devices (Janela > Dispositivos* ou *Janela > Organizador > Devices Xcode 5)*.
3. Clique no seu dispositivo.
4. Ao lado do rótulo identificador há uma sequência de caracteres:





**Figura 10 – UDID do dispositivo.**

## Criar e instalar um perfil de aprovisionamento

Perfis de aprovisionamento são usados para ligar um desenvolvedor e dispositivos para a equipe de desenvolvimento. Esse perfil de aprovisionamento é necessário para a execução de aplicativos em um dispositivo iOS.

Se você é um desenvolvedor individual, é necessário criar um perfil de aprovisionamento. Para obter informações específicas, consulte:

<https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppStoreDistributionTutorial/Introduction/Introduction.html>.

Depois que seu perfil de aprovisionamento for criado, você deve instalá-lo no Xcode.

Para o Xcode 5:

1. Abra o Xcode no Mac e vá para o *Organizer* (*Window > Organizer*).
2. Na seção “Biblioteca”, selecione “Perfis de aprovisionamento” e clique em “Atualizar”.
3. O Xcode pede que você entre com seu *Apple ID*. Insira suas credenciais e selecione “Iniciar sessão”.
4. Selecione um perfil válido, arraste-o e solte-o nos perfis de aprovisionamento de seu dispositivo de teste.

Para o Xcode 6:

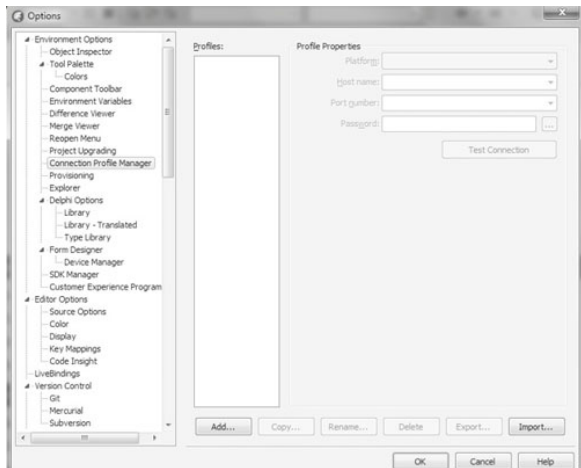
1. Abra o Xcode no Mac e vá para *Devices* (*Janela > Devices*). Com o botão direito do mouse no dispositivo, selecione “Visualizar perfis de aprovisionamento...”.
2. Clique no botão mais (+) para adicionar um perfil de aprovisionamento.
3. Encontre o seu perfil de aprovisionamento, selecione-o e clique em “Instalar”.
4. Clique em “Concluído” para fechar a janela.

## **Configurar o Delphi® para compilar em um dispositivo ou simulador iOS**

Após toda a configuração no Mac OS X, certifique-se de que o *Platform Assistant* esteja devidamente instalado e em execução em seu dispositivo Mac.

Agora, em seu computador PC, com Microsoft® Windows, execute a IDE do Delphi® e crie um novo projeto. Podemos trabalhar com qualquer tipo de projeto, para efeitos de teste.

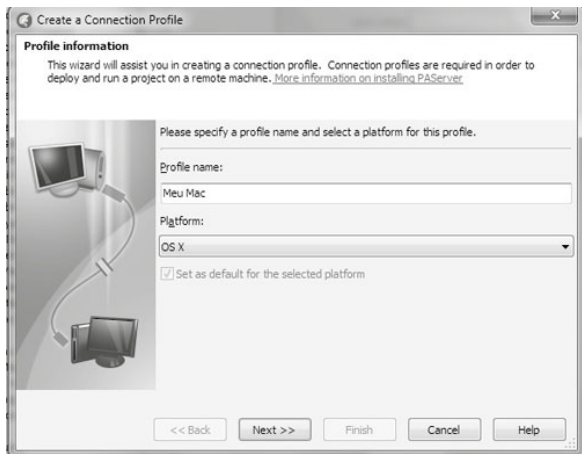
Com o Delphi® aberto, vá até *Tools > Options > Environment Options > Connection Profile Manager*.



**Figura 11 – Connection Profile Manager.**

Clique em “Add...”.

Agora você vê o assistente. Defina um nome para o perfil de conexão, como “Meu Mac”. Certifique-se de selecionar OS X como plataforma e, em seguida, clique em “Next >>”:



**Figura 12 – Assistente para criação de um novo perfil de conexão.**

Na página de informações da máquina remota, defina o nome ou endereço IP do *host* Mac, um número de porta para usar (a porta 64211, padrão, normalmente funciona) e uma senha opcional (se você quiser usar uma senha).

Clique em “Test Connection” e certifique-se de que o perfil de conexão foi bem-sucedido (você deve receber a mensagem “Connection to <hostname> on port <portnumber> succeeded”).

# 3. Configurando o Ambiente para Desenvolvimento Android

---

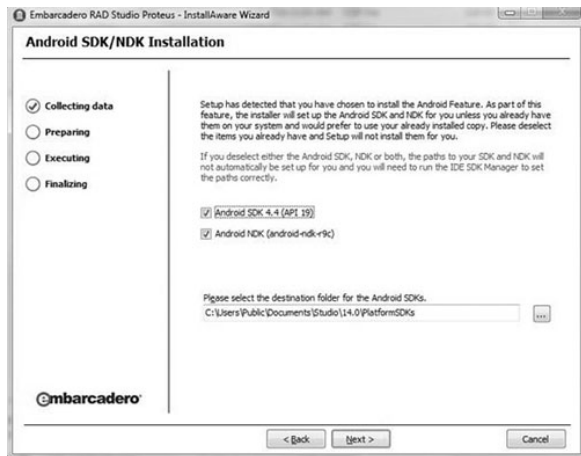
Sem taxa de adesão ou qualquer instalação ou contratos burocráticos, é possível executar, testar e até mesmo distribuir sua aplicação Android, pela internet ou através do cabo USB, conectado ao PC e ao dispositivo. Opcionalmente, é possível testar o aplicativo em um emulador Android. Você não precisará do *Platform Assistant*, uma vez que ele só é necessário para o *deploy* em dispositivos Mac OS X e iOS.

**Nota: se você instalar as ferramentas de desenvolvimento do Android durante a sua instalação do Delphi, instale o *driver* USB no seu dispositivo físico Android.**

As seguintes ferramentas de desenvolvimento do Android estão instaladas em seu sistema de desenvolvimento durante a instalação do Delphi:

- *Java Development Kit* (JDK).
- *Android Software Development Kit* (SDK).
- *Android Native Development Kit* (NDK).

Recomendo que você instale as ferramentas de desenvolvimento do Android usando o instalador do produto. No entanto, se você já possui as ferramentas de desenvolvimento do Android instaladas em seu sistema, você pode optar por pular essa etapa no instalador.



**Figura 13 – Instalação do Delphi®.**

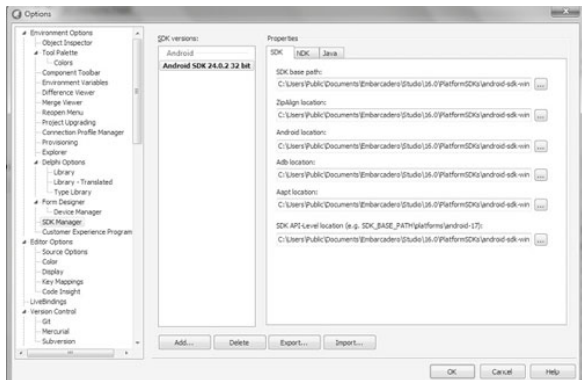
## Instalação do SDK e NDK do Android no RAD Studio

| Ferramenta                                    | Caminho  |
|---|--|
| <i>Java Development Kit (JDK)</i>             | C:\Program Files\Java\jdk1.7.0_25  |
| <i>Android Software Development Kit (SDK)</i> | C:\Users\Public\Documents\Embarcadero\Studio\15.0\PlatformSDKs\adt-bundle-windows-x86-20131030\sdk |
| <i>Android Native Development Kit (NDK)</i>   | C:\Users\Public\Documents\Embarcadero\Studio\15.0\PlatformSDKs\android-ndk-r9c                     |

Se o SDK e o NDK foram instalados durante a instalação do RAD Studio, o SDK do Android deve ser descoberto automaticamente pelo Delphi<sup>®</sup>, e você não precisa ler este capítulo.

Se você instalou o SDK e o NDK do Android manualmente, você precisa executar este passo para que o Delphi® possa criar aplicativos para os dispositivos Android.

1. Selecione *Tools > Options > Environment Options > SDK Manager*.
2. Clique em “Add...”. Uma caixa de diálogo “Add a New SDK” aparecerá. Clique na seta para baixo na lista, selecione uma versão SDK e selecione “Add New...” a partir da lista *drop-down*.



**Figura 14 – SDK Manager.**

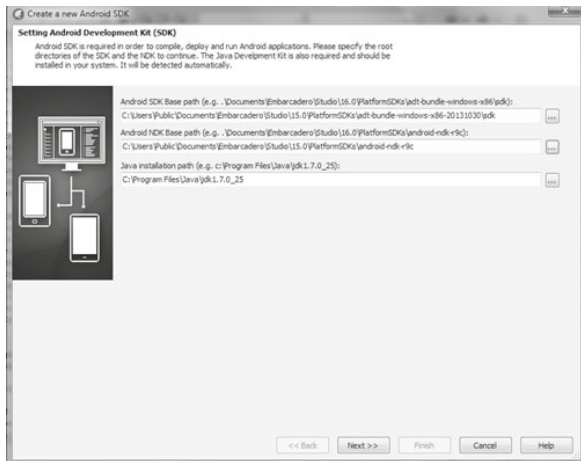


**Figura 15 – Add a New SDK.**

**Nota:** os campos ficam em branco no caso de o *SDK Manager* não conseguir detectar o padrão instalado e você ainda não tiver adicionado um SDK Android para o gerenciador de SDK.

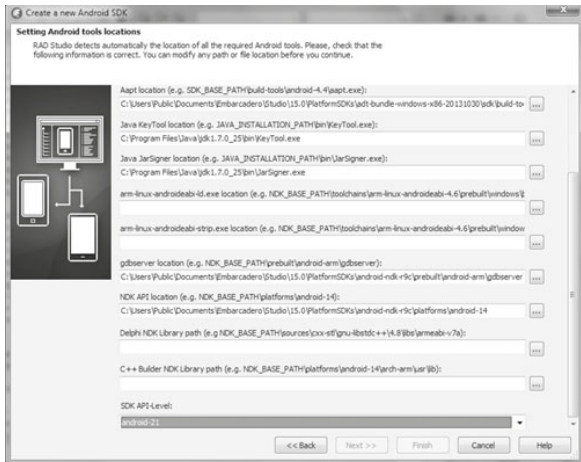
Complete os dois primeiros campos neste assistente: *Android SDK Base path* e *Android NDK Base path* (seus locais instalados). Você pode indicar os caminhos dos locais instalados ou clicar nas reticências (...) e navegar até os locais instalados.





**Figura 16 – Wizard Android SDK.**

Clique em “Next >>” para ir para a página seguinte do assistente, que tentará preencher os campos pelas escolhas.



**Figura 17 – Final do assistente.**

Para todos os campos que não foram preenchidos, clique no botão de reticências (...) e navegue até o local instalado.

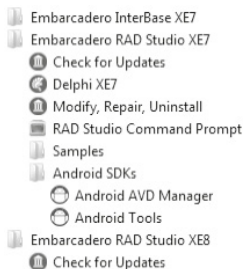
Clique em “Finish” para fechar o assistente.

## Instalar o controlador USB para o seu dispositivo Android

Instruções e sites de *download* estão disponíveis para vários dispositivos Android:

- <http://developer.android.com/sdk/win-usb.html> (Google USB Drivers).
- <https://developer.amazon.com/sdk/fire/connect-adb.html> (Kindle Fire USB Drivers)
- <http://developer.android.com/tools/extras/oem-usb.html> (OEM USB Drivers)

Inicie o SDK Manager.exe selecionando *Iniciar > Programas > Embarcadero RAD Studio > SDKs Android > Android Tools*:

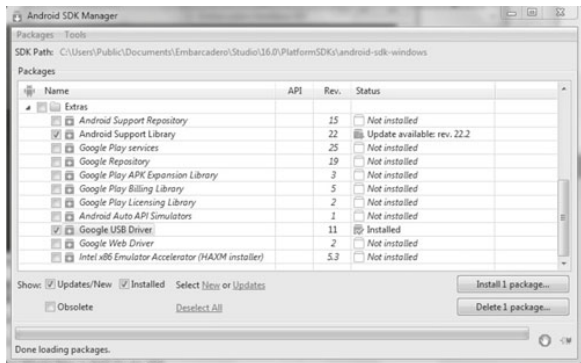


**Figura 18 – Android Tools.**

**Dica: se o gerenciador de SDK Android não iniciar, execute o arquivo `android.bat` do diretório `ferramentas` dentro do seu diretório do SDK instalado.**

No gerenciador de SDK Android, instale o *driver* USB para o seu dispositivo:

- Selecione *Google USB Driver*, desmarque todas as outras caixas de seleção e clique em “Install 1 package...”.



**Figura 19 – Instalar Google USB Driver.**

- Ligue seu dispositivo Android e conecte-o através de um cabo USB para no seu computador de desenvolvimento.
- No Painel de Controle, no Gerenciador de Dispositivos do seu sistema operacional, clique com o botão direito do mouse no seu aparelho e selecione “Atualizar driver”.
- Na caixa de diálogo “Atualizar driver”, navegue até a pasta \ extras em seu diretório SDK Android e selecione a pasta que corresponde ao seu dispositivo.
- Clique em “OK”.

## Ativando o modo de depuração USB no dispositivo

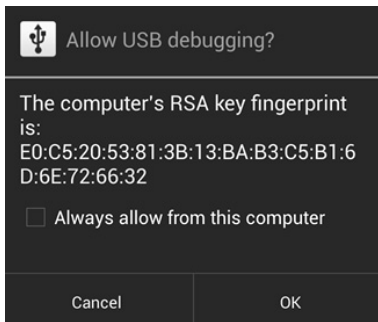
| Dispositivo Android                  | Instruções   |
|--------------------------------------|--|
| Android 4.0 ou superior (Jelly Bean) | Ative a opção “Depuração USB” em <i>Configurações &gt; Opções do desenvolvedor</i> .<br>Para Android 4.2 e mais recentes, opções de desenvolvedor estão ocultas por padrão. Use as seguintes etapas: |

No dispositivo, vá para *Configurações > Sobre <dispositivo>*.  
 Toque o número de compilação sete vezes para deixar *Configurações > Opções do desenvolvedor* disponível.  
 Em seguida, habilite a opção de depuração USB.  
**Dica: você também pode querer habilitar a opção “Permanecer acordado”, para evitar que o seu dispositivo Android “durma” enquanto conectado à porta USB.**  
**Se você estiver executando o Windows 8, você deve configurar a conexão USB como Câmera (PTP).**  
 Dependendo da versão do Android, você deve fazer um dos seguintes procedimentos:  
 Arraste a barra de status, toque em “Conectado como <tipo de conexão> Notificações” e selecione Câmera (PTP).  
 Vá para *Configurações > Armazenamento > Menu > Conexão USB do computador* e selecione Câmera (PTP).

Dispositivos  
Kindle Fire

Selecione *Configurações > Segurança* e definir “Ativar ADB” como “Ligado”.  
 Para mais informações, consulte a documentação do Amazon.

Se você estiver executando o Android 4.2.2 ou posterior, você pode ver a seguinte caixa de diálogo quando conectar o dispositivo ao PC executando o Delphi®:



**Figura 20 – Permitir depuração USB.**

Clique em “OK” para permitir a depuração USB no seu PC. Se, acidentalmente, você clicou em “Cancel”, desconecte o dispositivo do seu PC e conecte-o de volta para trazer a mensagem novamente no seu dispositivo.

## **Criando um novo emulador**

Um emulador Android permite que você execute uma aplicação Android em seu sistema de desenvolvimento, em vez de em um dispositivo Android. Um emulador pode ser útil durante os ciclos de desenvolvimento, mas normalmente tem problemas significativos de desempenho não vistos em dispositivos Android.

Um emulador Android é um dispositivo virtual (*Android Virtual Device* – AVD) que representa um dispositivo Android específico. Você pode usar um emulador Android como uma plataforma de destino para executar e testar suas aplicações Android em seu PC.

Emuladores Android são extremamente lentos e não suportam o mesmo conjunto de recursos como nos dispositivos reais.

Não recomendo que você tente usar um emulador por ser muito lento. A melhor maneira é testar em um dispositivo físico. De qualquer forma, caso você ainda não tenha condições de testar em um aparelho, deixo os passos a seguir para criar um emulador e executar seus aplicativos desenvolvidos no Delphi®.

Recomenda-se a seguinte configuração em seu PC:

- Intel Core i5 CPU.
- 4 GB de memória RAM.
- Emulação GPU.

O Delphi® suporta o emulador para Android versões 4.x. Emuladores para versões mais antigas não são suportados, pois são para processadores ARMv6.

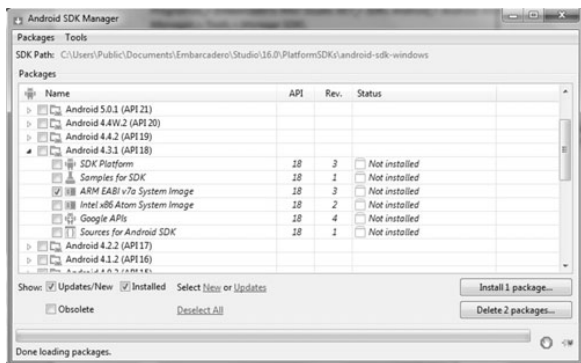
Antes de criar um emulador Android, você precisa instalar uma imagem do sistema Android através do SDK.

**Nota: se você instalou o SDK e o NDK Android durante a instalação do Delphi, uma imagem do sistema Android válido (como Android 4.2.2 API 17) já deve ser mostrada como instalada no gerenciador SDK do**

## Android. Nesse caso, você não precisa instalar uma outra imagem do sistema Android.

Para criar um emulador, siga esses passos:

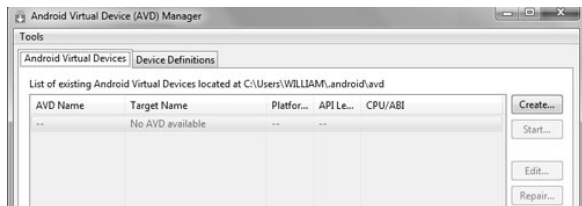
1. Inicie o *Android Virtual Device Manager* (selecione *Iniciar > Todos os Programas > Embarcadero RAD Studio XE7 > SDKs Android > Android AVD Manager > Tools > Manage SDK*).
2. Na árvore de pacotes, localizar e verificar a ARM EABI v7a dentro do primeiro nó na lista. Por exemplo: 4.2.2 Android (API 17) ou Android 4.3 (API 18).
3. Clique em “Install 1 package...”.
4. Na escolha de pacotes para a página de instalação, clique em “Accept License” e clique em “Install 1 package...” para instalar a imagem do sistema.



**Figura 21 – Instalação de uma imagem Android através do AVD Manager.**

Feche o SDK e inicie o *Android Virtual Device Manager* (selecione *Iniciar > Todos os Programas > Embarcadero RAD Studio XE7 > SDKs Android > Android AVD Manager*).

Clique no botão “New...” para criar um novo dispositivo virtual.



**Figura 22 – Repositório de emuladores Android.**

Selecione um dispositivo Android para emular e digite os detalhes descrevendo o dispositivo Android que você quer imitar. A fim de executar uma aplicação multidispositivo, o emulador Android deve atender aos seguintes requisitos:

- Em “Target”, selecione um SDK Android com um nível de API 19 ou superior. A lista *drop-down* contém as versões instaladas do SDK Android.
- Em “Emulation Options”, marque “Use Host GPU”.
- Em “Device”, selecione o dispositivo Android para emular.

No Delphi®, no *Project Manager*, o seu emulador deve aparecer em *Target Platform*, com a sua porta atribuída (5554):



**Figura 23 – Emulador na porta 5554.**



## Criando uma nova conta na Google Play

Antes de enviar seu aplicativo para a loja do Google, a *Google Play Store*, é necessário criar uma conta de desenvolvedor e, até o presente momento, pagar uma taxa única de adesão de 25 dólares americanos.

Para criar sua conta de desenvolvedor basta acessar o seguinte endereço:

<http://developer.android.com/index.html>

Você será redirecionado ao *Developer Console* e ao painel de controle do desenvolvedor para realizar o seu registro e, por fim, submeter seu aplicativo à loja.

Crie sua conta, faça o pagamento da taxa e volte ao Delphi<sup>®</sup>.

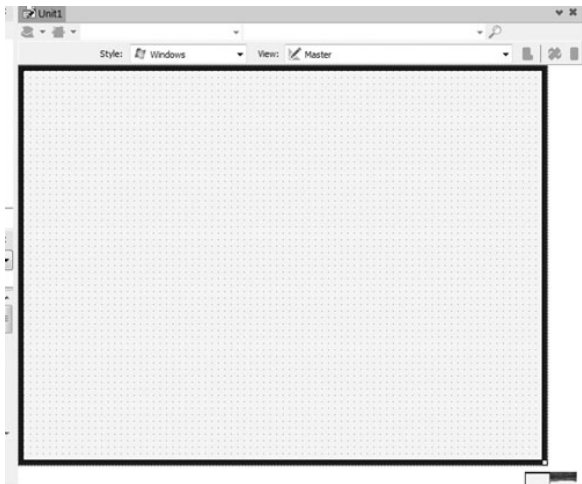
## 4. Hello World

---

Após realizar as configurações e selecionar sua plataforma preferida para o desenvolvimento, chegou a hora de você iniciar os seus projetos. Para não fugir do clássico, aqui vai um “Hello World”.

Inicie sua IDE do Delphi<sup>®</sup>, abra um novo projeto *Multi-Device* e selecione *blank application*.

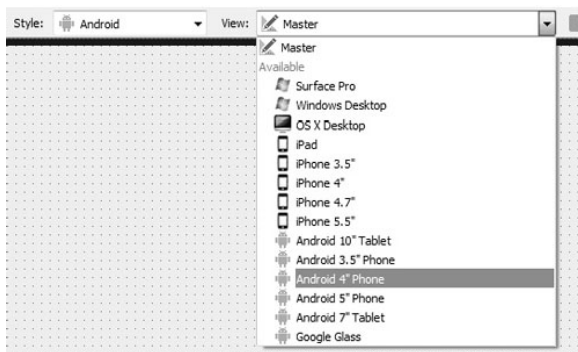
A tela principal, contendo um formulário, será apresentada conforme a figura.



**Figura 24 – Tela principal.**

Observe que, a partir do Delphi® XE7, a forma de desenhar formulários segue um novo padrão, chamado de *Fire UI*, onde é possível desenvolver um único formulário para diversos dispositivos. E, é claro, com design responsivo.

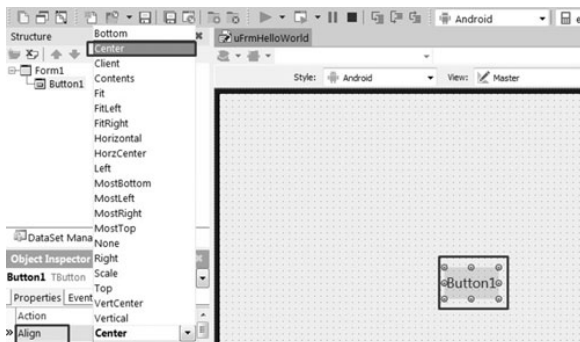
Em “Style”, modifique para Android e opte por enquanto para trabalhar no *view Master*. Falarei sobre as outras formas de visualização, porém, antes, é necessário que você entenda o principal objetivo de equilibrar os componentes e deixar sua tela responsiva.

**Figura 25 – Fire UI em ação.**

Qualquer componente visual inserido em seus formulários deve estar no *view Master*, isso é mandatório. O que é possível fazer é trocar a posição, cor e outras propriedades visuais em modo de projeto para cada visualização disponível.

**Nota: lembre-se de marcar no *Project Manager* a plataforma de destino para que o visual do *Form* seja alterado também em modo de projeto. Por padrão, o Delphi® configura a plataforma atual em um novo projeto como Windows 32-bits.**

Insira um botão do tipo *TButton* (sim, o mesmo *TButton* clássico, desde o Delphi<sup>®</sup> 1) em seu formulário e altere a sua propriedade *Align* para o valor *Center*.



**Figura 26 – TButton com alinhamento no centro.**

Altere a visão em *Views* para “Android 4” Phone” e veja que o botão está **exatamente** no centro da tela. Portanto, lembre-se de trabalhar sempre na *view master* e realizar seus ajustes finos nas visualizações das plataformas desejadas.

Dê um duplo clique no botão e será direcionado ao evento *OnClick*. Preencha com o seguinte código:

```
procedure TForm1.OnButton1Click(Sender : TObject);
begin
  ShowMessage('Hello World');
end;
```

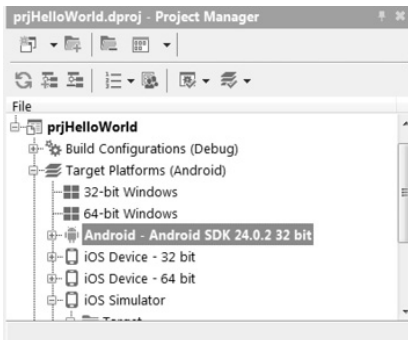
A implementação dessa aplicação está acabada, então agora você pode executá-la.

Você pode clicar no botão “Run” no IDE, pressionar F9 ou selecionar *Run > Run* no menu principal do Delphi<sup>®</sup>:

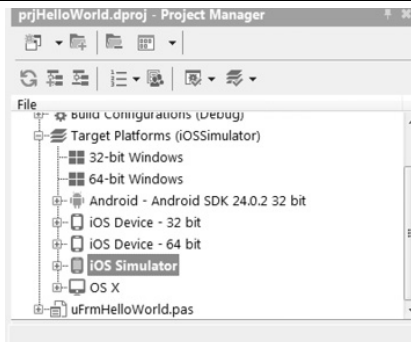


**Figura 27 – Executar sua aplicação.**

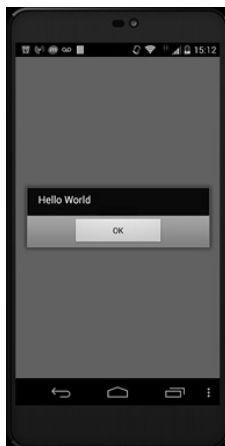
**Nota: antes de executar sua aplicação, certifique-se de que você selecionou a plataforma desejada, através do *Project Manager*.**



**Figura 28 – Plataforma Android selecionada.**



**Figura 29 – Plataforma iOS selecionada.**



**Figura 30 – Exemplo em execução.**

## 5. Componentes Básicos para Interface Visual

---

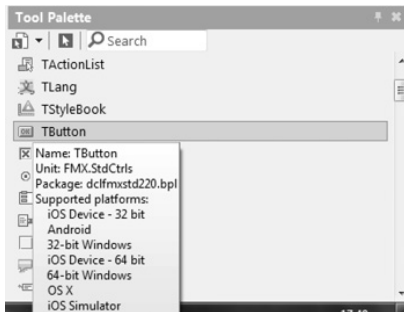
Os componentes visuais nativos do FireMonkey foram desenhados para o funcionamento de diferentes formas nas plataformas suportadas, adequando-se ao melhor estilo esperado para as plataformas Windows, Mac OS X, Android e iOS.

É muito simples trabalhar com os estilos predefinidos da IDE do Delphi®, uma vez que é possível ainda efetuar o *download*, através do *Embarcadero Developer Network*, de estilos adicionais, personalizando ainda mais a aparência de suas aplicações *mobile*.

### Uso de componentes nativos

O Delphi®, de forma nativa, define o melhor visual para determinada plataforma de desenvolvimento. Se estiver compilando para Windows, ele deixará seu formulário e seus componentes o mais padronizado possível com o Windows. O mesmo ocorre com as outras plataformas, como é o caso do Android e iOS.

Um jeito bastante prático de descobrir se um componente é compatível com a plataforma de destino desejada é deixar o mouse sobre o nome do componente, na *Tool Palette*, até que apareça, através de um *Hint*, quais plataformas são suportadas.



**Figura 31 – Plataformas para *TButton*.**

Repare que, para um componente visual do tipo *TButton*, todas as plataformas disponíveis para compilação no Delphi<sup>®</sup> são suportadas.

## Botões com diferentes estilos

O FireMonkey define vários tipos de botões, que você pode utilizar com os mesmos passos descritos aqui. Os botões incluem *TButton* e *TSpeedButton*.



**Figura 32 – Botões no iOS.**

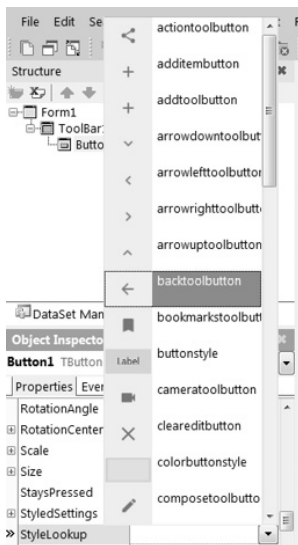


**Figura 33 – Botões no Android.**

Para modificar o estilo predefinido de um botão ou de qualquer outro componente, basta alterar a propriedade *StyleLookup*, através do *Object Inspector*.

Na lista *drop-down*, você pode selecionar um estilo predefinido com base em como o componente deverá ser utilizado.





**Figura 34 – StyleLookup de um TButton.**

**Nota:** para criar um botão colorido, altere os valores das propriedades *tintColor* e *IconTintColor*. A última propriedade está disponível apenas para os botões decorados com ícones.

- *TintColor* especifica a cor de fundo do botão.
- *IconTintColor* especifica a cor do ícone em botões de estilo.

As propriedades *tintColor* e *IconTintColor* só estão disponíveis no *Object Inspector* se você selecionar um estilo adequado para o botão e escolher uma exibição adequada no *Form Designer* (essas propriedades não são visíveis em todas as *views*), como, por exemplo, no *Windows*.

## Para o Android

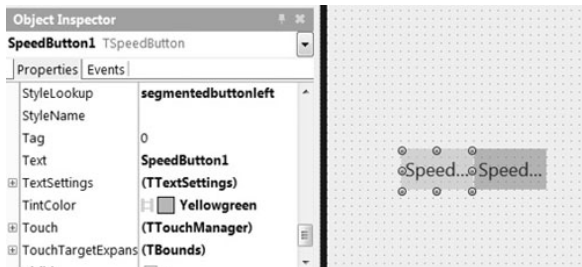
Você pode aplicar uma tonalidade para a maioria dos botões de qualquer estilo:

- Para os *Speed Buttons*, você precisa selecionar um valor *StyleLookup* adequado, a fim de alterar o valor *tintColor* no *Object Inspector*.
- Para modificar *tintColor* e *IconTintColor*, escolha um dispositivo Android a partir da lista de *views*, no canto superior direito.

## Para o iOS

- O FireMonkey fornece botões que correspondem ao “Guia de Estilo da Apple”, e alguns botões podem não suportar o recurso de tonalidade.
- Por exemplo, no iOS 7, botões segmentados têm a propriedade *tintColor* (No iOS 6, os botões segmentados não suportam o recurso de tonalidade).

Quando você alterar a propriedade *StyleLookup* de um botão, o *Object Inspector* exibe automaticamente ou oculta as propriedades *tintColor* e *IconTintColor*, conforme o caso.



**Figura 35 – Botões segmentados (*SegmentedButton*).**

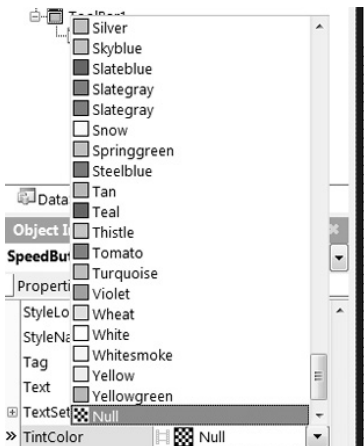


Figura 36 – Seleção de cor do *SpeedButton*.

## Layouts responsivos – Diferentes telas, diferentes dispositivos

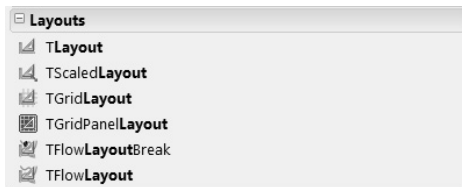
Você deve saber que o padrão Delphi<sup>®</sup> para desenvolvimento visual consiste em botões, caixas de edição para texto, caixas de combinação ou seleção, checagem, entre outros.

Todos os componentes básicos estão presentes, porém, dependendo da plataforma selecionada, nem todos terão o mesmo destaque visual. Não é necessário desenhar um formulário diferente para cada tipo de dispositivo ou tamanho de tela. Para contornar esse problema, utilizamos o **design responsivo**.

O Delphi<sup>®</sup> vem equipado com alguns componentes que, junto com o recurso de *Views*, deixam o visual responsivo e elegante de acordo com o dispositivo.

O componente *TLayout* é um exemplo de componente não visual, responsável por agrupar outros componentes e direcionar seu alinhamento de forma que

qualquer dispositivo respeite suas dimensões padronizadas, remetendo, assim, aos visuais “ancorados” em aplicações *desktop* (quando o usuário maximiza a janela, por exemplo).



**Figura 37 – Componentes de Layout.**

Tipos de layout:

- **TLayout** – Um *container* simples que não é visível em modo de execução e pode ser utilizado para agrupar outros controles para serem manipulados como um todo. Por exemplo, você pode definir a visibilidade de um grupo de controles ao mesmo tempo, definindo apenas a propriedade *Visible* do layout.
- **TScaledLayout** – Um *ScaledLayout* é um *container* que oferece a possibilidade de dimensionar um grupo de objetos gráficos de acordo com as dimensões físicas do layout. Os controles filhos esticam junto com o layout quando este é redimensionado. *TScaledLayout* mantém seu tamanho original através das propriedades *OriginalWidth* e *OriginalHeight*.
- **TGridLayout** – Organiza controles filhos em uma grade de células de tamanho igual. Ao usar o *TGridLayout*, os controles filhos são redimensionados para caber nas dimensões especificadas através das propriedades *ItemHeight* e *ItemWidth*. As propriedades de altura e largura para os controles filhos são automaticamente definidas.
- **TGridPanelLayout** – Organiza controles filhos em um painel do *grid*. Cada controle filho é colocado dentro de uma célula em um painel do *grid*. Ao contrário do *TGridLayout*, você pode redimensionar manualmente e alinhar os controles filhos colocados no interior das células. Ou seja, usando o layout *TGridPanelLayout*, você pode definir explicitamente as propriedades de

altura e largura de controles filhos. Você também pode definir explicitamente as propriedades *Align* e *Anchor*s dos controles filhos.

- **TFlowLayout** – Organiza seus controles filhos como se fossem palavras em um parágrafo. Ao usar um *TFlowLayout*, os controles filhos são organizados e exibidos no layout, seguindo a ordem em que foram adicionados. Para iniciar a exibição de controles filhos em uma nova linha, adicione um *TFlowLayoutBreak*. O comportamento resultante é como se fosse uma nova linha de um parágrafo de um texto. *TFlowLayout* segue um conjunto de regras para organizar os seus filhos. Essas regras podem ser personalizadas através destas propriedades: *justify*, *JustifyLastLine*, *verticalGap* e *horizontalGap*. Por padrão, os controles adicionados a um *TFlowLayout* após um *TFlowLayoutBreak* estão dispostos a seguir as regras especificadas pelo *TFlowLayout*. Esse comportamento pode ser alterado, definindo *ChangesRules* da classe *TFlowLayoutBreak* para *True*. Se *ChangesRules* é definido como *True*, os controles filhos adicionados após a *TFlowLayoutBreak* são posicionados de acordo com as regras especificadas através das propriedades *TFlowLayoutBreak*, até que todos os controles que estão sendo exibidos encontrem outro *TFlowLayoutBreak*.

## Criando layouts com *Scroll*

A criatividade e a perspicácia de um designer em aplicações *mobile* são testadas a todo instante. A cada dia surgem novos aparelhos, e com diferentes tamanhos de tela. São *tablets* e *smartphones* com 4, 5, 6 e até 7 polegadas. Às vezes, desenhar uma tela de cadastro com um pouco mais de cinco ou seis campos para preenchimento pode dificultar a vida, pois será necessário que o usuário “role para baixo” a tela, através do *touch screen*.

Alguns componentes nativos do Delphi<sup>®</sup> já vêm com essa funcionalidade de *scroll* automático, como por exemplo o *ListBox* e o *ListView*. Serve para telas em que são construídos formulários puros de cadastro, com diversos campos. Nem sempre o conteúdo caberá em um dispositivo de 3,5 polegadas, por exemplo. Para isso existem os componentes de *Scroll*.

## Scroll layout: TScrollBox

Um *scroll layout* oferece a possibilidade de deslocar grupos de objetos gráficos. Você pode usar caixas de rolagem para criar áreas de rolagem dentro de um formulário. Essas áreas são conhecidas como **pontos de vista**. Visualizações são comuns em aplicações de processador de texto, planilha e gerenciamento de projetos comerciais. Para delimitar a área de rolagem, utilize *TFramedScrollBox*.

Na classe *TScrollBox*, a propriedade *AniCalculations* manipula o objeto tipo *TAniCalculations*, que fornece suporte para a rolagem inercial suave.

O código a seguir é o mais interessante.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    //Edite essas propriedades para alterar os parâmetros de scroll
    ScrollBox1.AniCalculations.TouchTracking := [ttVertical, ttHorizontal]; //
    [ttVertical]; //
    ScrollBox1.AniCalculations.Animation := True; // False; //
    ScrollBox1.AniCalculations.BoundsAnimation := False; // True; //
    ScrollBox1.AniCalculations.DecelerationRate := 0.5; //Decelera-
    tionRateFast;
    //DecelerationRateNormal; //
    ScrollBox1.AniCalculations.AutoShowing := True; // False; //
end;
```

No código, no *FormCreate*, altere as propriedades do seu *scrollbox* para ativar as animações necessárias para que os eventos de deslizar com os dedos, através do *touch screen*, surtam efeito.

Um outro componente, o *TVertScrollBox*, completa a solução responsiva, junto com o pacote de *Views* e layouts, para suas aplicações *mobile*. Parece que isso é uma funcionalidade padrão, porém não é. Imagine digitar em seu editor de e-mail, por exemplo, e o teclado virtual sobrepôr a caixa de texto! Chato, não? Para resolver esse problema, utilize o componente *TVertScrollBox*, junto com os eventos de seu *Form*: *OnVirtualKeyboardShown*, *OnVirtualKeyboardHidden* e *OnFocusChanged*.

Use o evento *OnVirtualKeyboardShown* para detectar se o teclado virtual aparece. Além disso, você pode recuperar as coordenadas do teclado virtual a partir desse evento.

O evento *OnVirtualKeyboardHidden* ocorre quando o teclado virtual desaparece.

O evento *OnFocusChanged* ocorre quando o foco está ajustado para outros controles.

Para criar um efeito de “*form* deslizante”, codifique os eventos, como descrito a seguir.

Crie duas variáveis globais, para controle dos limites da tela. Chamemos de *FNeddOffset*, do tipo *Boolean*, e *FKBBounds*, do tipo *TRect*. Inclua no *uses* do seu projeto a *unit System.Math*.

Crie duas *procedures* com o nome, por exemplo, de “AtualizaLimites” e “RestaurarPosicao”.

```
procedure TFormMain.AtualizaLimites;
var
  LFocused : TControl;
  LFocusRect: TRectF;
begin
  FNeedOffset := False;
  if Assigned(Focused) then
    begin
      LFocused := TControl(Focused.GetObject);
      LFocusRect := LFocused.AbsoluteRect;
      LFocusRect.Offset(VertScrollBar1.ViewportPosition);
      if (LFocusRect.IntersectsWith(TRectF.Create(FKBBounds))) and
        (LFocusRect.Bottom > FKBBounds.Top) then
        begin
          FNeedOffset := True;
          MainLayout1.Align := TAlignLayout.Horizontal;
          VertScrollBar1.RealignContent;
          Application.ProcessMessages;
          VertScrollBar1.ViewportPosition :=
```

```

        PointF(VertScrollBar1.ViewportPosition.X,
            LFocusRect.Bottom - FKBBounds.Top);
    end;
end;
if not FNeedOffset then
    RestaurarPosicao;
end;

```

```

Procedure TFormMain. RestaurarPosicao;
begin
    VertScrollBar1.ViewportPosition := PointF(VertScrollBar1.ViewportPosition.X, 0);
    MainLayout1.Align := TAlignLayout.Client;
    VertScrollBar1.RealignContent;
end;

```

O evento a seguir é para quando surgir o teclado virtual – esse evento pertence ao formulário.

```

procedure TFormMain.FormVirtualKeyboardShown(Sender: TObject;
    KeyboardVisible: Boolean;
    const Bounds: TRect);
begin
    FKBBounds := TRectF.Create(Bounds);
    FKBBounds.TopLeft := ScreenToClient(FKBBounds.TopLeft);
    FKBBounds.BottomRight := ScreenToClient(FKBBounds.BottomRight);
    AtualizarLimites;
end;

```

Este é o evento para quando o teclado virtual for desativado, ou seja, quando o usuário optar por não mais utilizá-lo:

```

procedure TFormMain.FormVirtualKeyboardHidden(Sender: TObject;
    KeyboardVisible: Boolean;
    const Bounds: TRect);
begin

```



```
FKBBounds.Create(0, 0, 0, 0);
FNeedOffset := False;
RestaurarPosicao;
end;
```

Programar o evento *OnCalcContentBounds* do seu *VertScrollBar*:

```
procedure TFormMain.VertScrollBar1CalcContentBounds(Sender: TObject;
  var ContentBounds: TRectF);
begin
  if FNeedOffset and (FKBBounds.Top > 0) then
    begin
      ContentBounds.Bottom := Max(ContentBounds.Bottom,
        2 * ClientHeight - FKBBounds.Top);
    end;
end;
```

Programar o evento *FormFocusChanged* do seu formulário:

```
procedure TFormMain.FormFocusChanged(Sender: TObject);
begin
  AtualizarLimites;
end;
```

**Nota:** para um perfeito funcionamento, você deve alterar uma variável global através do arquivo *projeto.dpr* em seu projeto. Para isso vá até o nome do projeto no *Project Manager* e, com o botão direito do mouse, clique em *View Source*. Estando com a *unit* do seu *projeto.dpr* em modo codificação, inclua depois do *Application.Initialize* o trecho a seguir.

```
Begin
  Application.Initialize;
  VKAutoShowMode := TVKAutoShowMode.Always;
  //Segue o fluxo....
```

```
End;
```

Uma outra dica é manter seu formulário em modo retrato ou paisagem, ou seja, o usuário poderá ou não ter um layout diferente ao mover o telefone, como você já faz para visualizar melhor aquela foto bacana do Facebook. Também no seu arquivo *projeto.dpr*:

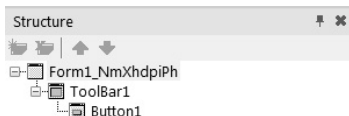
```
Begin
  Application.Initialize;
  VKAutoShowMode := TVKAutoShowMode.Always;
  Application.FormFactor.Orientations := [TFormOrientation.Portrait];
  //Segue o fluxo...
End;
```

Para maiores detalhes sobre este exemplo, visite:

[http://docwiki.embarcadero.com/CodeExamples/XE7/en/FMX.Mobile.ScrollableForm\\_Sample\\_\(Delphi\)](http://docwiki.embarcadero.com/CodeExamples/XE7/en/FMX.Mobile.ScrollableForm_Sample_(Delphi)).

## Utilizando botões em barra de ferramentas

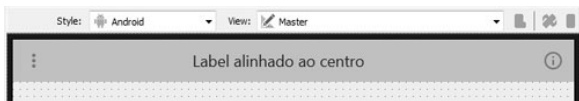
Em aplicativos *desktops* é muito comum a utilização de menus, principalmente os de estilo Notepad, Word, etc. Porém, o mesmo visual não é intuitivo e nem um pouco prático quando se trata de aplicativos móveis. Pensando nisso, você provavelmente utilizará os recursos do *TToolBar*, componente para desenvolvimento de barras de ferramentas.



**Figura 38 – Botão dentro de uma barra de ferramentas visto através do painel de estrutura.**

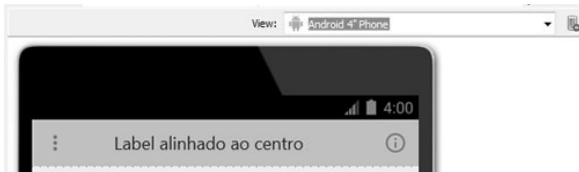
Certifique-se de que o botão ou outro controle adicionado ao formulário esteja contido no *Toolbar*. Você poderá visualizar a hierarquia dos componentes visuais através do painel *Structure*, no canto superior esquerdo da tela.

Para efeito de teste, crie uma *Toolbar* em seu formulário alterando o *Style* para Android ou iOS, porém mantenha a *View* como *Master*. Crie dois botões e alinhe-os para a esquerda e para a direita, respectivamente. Em seguida, adicione um *Label*, mas mude sua propriedade *Align* para *Center*. Em *TextSettings*, altere a subpropriedade *HorzAlign* para *Center*.



**Figura 39 – Toolbar com dois botões e um texto.**

Para visualizar o resultado e enfim vislumbrar seu design responsivo, altere a *View* para a plataforma desejada.



**Figura 40 – Toolbar em modo de visualização para Android de 4 polegadas.**

## Utilizando texto com teclados diferentes

Uma das facilidades que o usuário de dispositivos móveis deseja é digitar somente o necessário, com o mínimo de erros possível e atingir o resultado esperado. Sabendo que o tamanho de uma tela pode variar de 3,5 até 6 polegadas para um *smartphone* e de 7 até 11 ou 12 polegadas, dependendo do *tablet*, ter informações demais em um teclado virtual é um problema.

Pensando nisso, o Delphi<sup>®</sup> propôs uma solução simples e prática: através dos controles *TEdit*, é possível configurar como o teclado deve aparecer.

Exemplifico os tipos de teclado virtual que seu *TEdit* suporta, tanto para Android como para iOS.



**Figura 41 – KeyboardType para Android.**



**Figura 42 – KeyboardType para iOS.**

Você pode perceber que, para cada tipo de teclado, existe uma forma de visualização distinta. Por exemplo: para utilizar o teclado de discagem, o recomendável é o *PhonePad*, uma vez que apresenta, além dos números básicos, teclas para código de área como o (+) e teclas de função para serviços, como o (\*).

A simplicidade vem quando a única propriedade que você precisa alterar para o funcionamento de um teclado distinto do padrão é a **KeyboardType**, através do *Object Inspector*.

**Nota:** uma dica muito importante é que você pode alterar o desenho da tecla de retorno. Basta alterar a propriedade *ReturnKeyType*, onde é possível, por exemplo, definir como “Next” – e, nesse caso, o sistema operacional do dispositivo se encarregará de modificar o estilo da tecla de retorno, modificando-o de seta “->” para a palavra “Próx.”.

**Nota:** para esconder o teclado virtual sem que seja necessário o usuário utilizar as teclas de retorno do próprio dispositivo, basta alterar a propriedade *KillFocusByReturn* para *True*. Sendo assim, ao teclar na palavra “Prox.”, por exemplo, o teclado virtual desaparece e você poderá, eventualmente, direcionar o foco para outro componente ou iniciar qualquer outro tipo de evento programado.

## Criando menus com *TMultiView*

Desde o início da revolução dos dispositivos móveis, quando a Apple lançou o iPhone, iPad e iPodTouch, os designers vêm trabalhando como nunca para dar a melhor experiência visual possível aos seus usuários. E isso, é claro, não é só trabalho dos designers da Apple!

Pensando em melhorar o *user experience*, o famoso UX, o Delphi<sup>®</sup> vem nativo com um belíssimo componente para você desenvolver menus profissionais sem precisar de muito esforço. Este é o caso do *TMultiView*.

O *TMultiView* representa um *container* (painel *master*) para outros controles e fornece uma maneira para que você possa facilmente apresentar pontos de vista alternativos de informação.

O componente *TMultiView* permite implementar uma interface de visão mestre (*Master-Detail*), que pode ser usado para qualquer plataforma de destino (como um iOS, Android, ou aplicação *desktop*).

No painel mestre, você pode exibir uma coleção de todos os controles visuais, tais como caixas de edição, rótulos, tabelas e assim por diante.

O painel de detalhe exibe informações com base nas propriedades de controles do painel de mestre.

Um exemplo de uso, visualmente falando, é o menu do Facebook, com o padrão *Drawer*, disponível no Delphi<sup>®</sup>.

## Modos de apresentação

| Modo                  | Apresentação                        |
|-----------------------|-------------------------------------|
| <b><i>Drawer</i></b>  | Estilo Facebook (empurrar/sobrepôr) |
| <b><i>Panel</i></b>   | Painel acoplado                     |
| <b><i>Popover</i></b> | Menu estilo <i>pop-up</i>           |
| <b><i>Custom</i></b>  | Customizável                        |

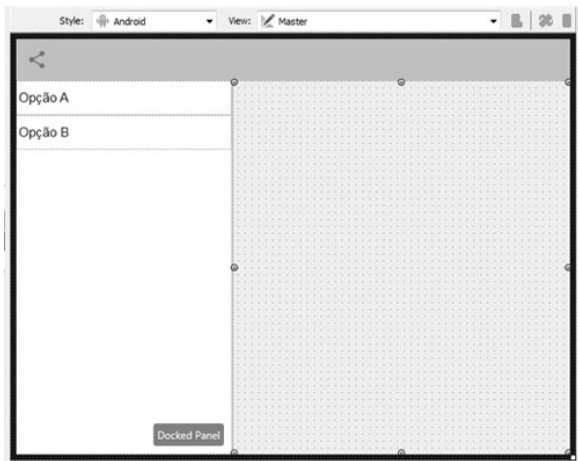
**Nota: no modo *custom* é necessário estender o evento *CustomPresentationClass* do seu *MultiView*.**

Adicione ao formulário antes um *Toolbar* e um *TButton* ou *TSpeedButton* como o botão que acionará o menu. Modifique o estilo do botão – a preferência, nesse caso, é sua.

Para trabalhar com um *MultiView*, vá até o *ToolPalette* e encontre-o em *Common Controls*.

Agora você pode adicionar um *TListBox* dentro do painel *master* e criar dois itens na lista, com os nomes “Opção A” e “Opção B”.

Por fim, adicione um componente do tipo *TLayout* na área de detalhes.



**Figura 43 – Formulário com menu *MultiView*.**

Agora você irá alterar algumas propriedades, a fim de transformar este em um menu estilo Facebook (*Drawer*). Para isso siga os passos:

1. Com o *MultiView* selecionado, altere a propriedade *MasterButton*, direcionando para o botão que você criou na *Toolbar*.
2. Ainda com o *MultiView*, altere a propriedade *TargetControl* para o seu componente de layout.
3. Você deve ter percebido que o botão ficou invisível. Para resolver e finalizar a edição, altere a propriedade *Mode*, ainda no *MultiView*, para *Drawer*.
4. Salve e execute.

O resultado será um formulário simples, com um botão no topo de uma barra de ferramentas. Porém, experimente clicar ou, no caso, tocar no botão para visualizar o menu estilo Facebook em ação.

**Nota: é possível modificar a cor de fundo e a transposição da animação dos menus, entre outras propriedades. Experimente.**



**Figura 44 – Menu estilo *drawer* em ação.**

**Nota: expanda o nó *ShadowOptions* e defina a propriedade *cor* para azul (esta propriedade define a cor da sombra do painel mestre. Você pode usar qualquer cor disponível).**



## Browser – Componente para navegação web

Você sabia que em uma de suas versões mais recentes o aplicativo do Facebook implementou um *browser* próprio, com opções de abertura em seu navegador padrão, envio por mensagem privada do link, entre outros?

O FireMonkey vem com um componente, baseado no projeto *Chromium*, disponível apenas para as plataformas *mobile*. Ou seja, você não poderá utilizar o *TWebBrowser* no Windows, por exemplo.

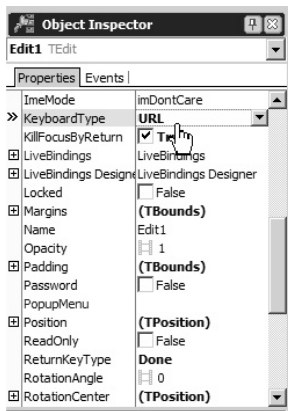
Para criar seu aplicativo contendo um navegador, basta inserir na tela o componente *TWebBrowser*, adicionar um *Toolbar*, um *TEdit* e um botão. Pronto! Já é o suficiente para você ter o mínimo necessário.

Para navegar por uma página web, é necessário que você codifique uma linha:

```
Procedure Navegar;  
Begin  
    WebBrowser1.Navigate(Edit1.Text);  
End;
```

Você pode criar uma *procedure* com o nome de **Navegar** e executá-la, por exemplo, no evento *OnChange* do seu *Edit1*.

Para um melhor aproveitamento, lembre-se de modificar a propriedade **KeyboardType** do seu *Edit1* para *URL* e defina o **ReturnKeyType**, por exemplo, para **Go**.



**Figura 45 – Teclado com formato para URL.**

Dentro do *Toolbar*, você pode programar o seu botão com o seguinte código:

```
Procedure TFrmPrincipal.TBtnVoltarOnClick(Sender : TObject);
Begin
  WebBrowser1.GoBack;
End;
```

Salve seu aplicativo e execute-o na plataforma desejada.



**Figura 46 – Componente *WebBrowser* em ação.**

## 6. *ListView* e *LiveBindings*

---

Não é comum nem usual utilizarmos componentes no estilo *Grid* para apresentação de dados em aplicativos móveis. Exemplo disso é a ausência de suporte ao componente *TDBGrid* para as plataformas Android e iOS.

Em se tratando de aplicativos móveis, apresentação de dados e até mesmo listas, nada mais vantajoso do que utilizar o componente *TListView*. Ainda mais com suporte *LiveBindings*!

Para efeito de testes, você irá aprender agora como manipular os eventos do *ListView* utilizando *LiveBindings*.

### Iniciando o projeto

Crie um novo projeto, *Blank Application*, e adicione os seguintes componentes no seu formulário principal:

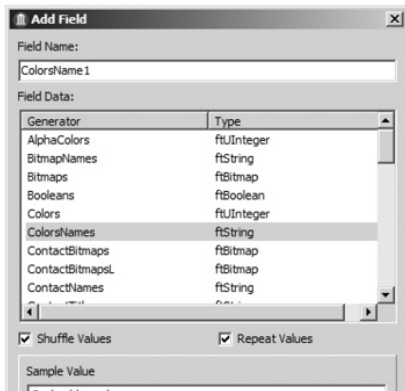
1. *TListView*, podendo ser localizada através do *ToolPalette*.
2. *TPrototypeBindSource*, também através do *ToolPalette*.

Vá no seu componente *ListView1* e modifique a propriedade *Align* para “Cliente” e *SearchVisible* para “True”.

### Adicionando campos

Selecione o componente adicionado do tipo *TPrototypeBindSource* e clique com o botão direito do mouse. Selecione a opção “Add Field...”.

Na caixa de diálogo que aparece, selecione o campo *ColorsNames*.

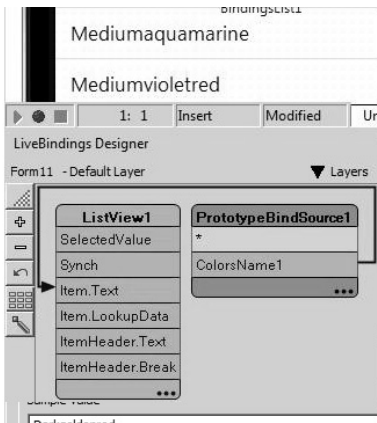


**Figura 47 – Caixa para adicionar campos.**

**Nota:** apenas a título de informação, o componente *TPrototypeBindSource* não serve para manipulação de dados reais, e sim apenas para o desenvolvimento de protótipos, como nesse caso. Utilize-o como uma prova de conceito para suas aplicações.

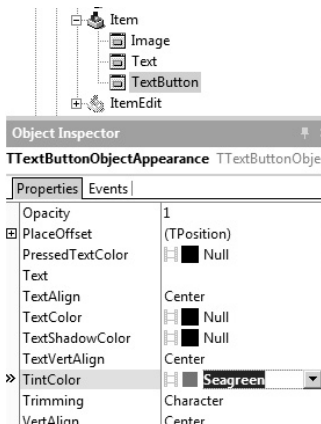
## Manipulando o *LiveBindings*

1. Abra o *LiveBindings Designer*. Vá no menu principal do Delphi® em *View > LiveBindings Designer* e arraste a propriedade *ColorsName1* do *TPrototypeBindSource* na propriedade *Item.Text* do *ListView* para vincular essas propriedades. O componente *ListView* preenche automaticamente os seus itens com nomes de cores a partir do componente de dados.



**Figura 48 – LiveBindings em ação.**

2. Defina a propriedade *TListView.ItemAppearance* para *ImageListItemRightButton*.
  - Coloque o foco no componente *ListView*, selecionando-o.
  - Em seguida, no *Object Inspector*, localize a propriedade *ItemAppearance* e altere seu valor para *ImageListItemRightButton*.
3. Opcionalmente, você pode aplicar uma tonalidade para os botões de texto.
  - Colocar o foco no componente *ListView*, selecionando-o.
  - No *Structure View*, sob *ItemAppearance*, expanda-o e, em seguida, selecione *TextButton*.
  - No *Object Inspector*, localize a propriedade *tintColor* e defina seu valor como *Seagreen*.



**Figura 49 – TextButton selecionado pelo Structure View.**

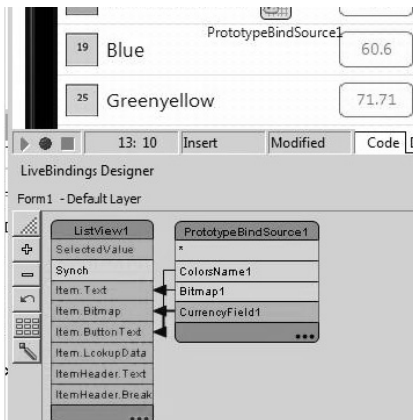
**Nota:** em modo de desenho, a cor que você aplicou aos botões de texto podem não ser visíveis. Para que as alterações se tornem visíveis, escolha o modo de exibição *master* no seletor de estilo para alterar o estilo atual do seu *Form Designer* para Android ou iOS.

## Adicionando mais campos

Para exibir uma imagem e um texto sobre o botão associado a cada item da lista, você precisa adicionar mais dois campos.

1. Com o botão direito do mouse sobre o componente *TPrototypeBindSource*, selecione “Add Field...”.
2. Na caixa de diálogo, dê um “Ctrl + Clique” para selecionar os campos de *bitmaps* e o campo *Currency*. Quando terminar, clique em “OK”.

3. Repita o processo do *LiveBindings*, como feito com o nome da cor, para vincular os campos de *Bitmap* e de moeda (*Currency*), nos *Item.Bitmap* e *Item.ButtonText*, respectivamente.



**Figura 50 – LiveBindings, novamente.**

Agora o *ListView* exibe os dados de moeda como texto do botão e os de cores com a cor propriamente dita.

## Adicionando o evento *onButtonClick*

Para criar uma aplicação prática, você pode manipular os eventos dos componentes e facilitar seu uso, como o caso do *onButtonClick*.

1. Selecione o componente *ListView1* e, na aba *Events*, procure por *onButtonClick*.

```
procedure TForm1.ListView1ButtonClick(const Sender: TObject;
const AItem: TListItem; const AObject: TListItemSimpleControl);
begin
```



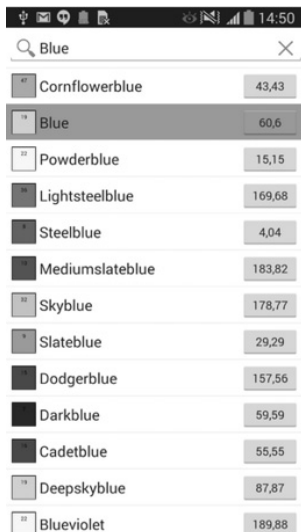
```
ShowMessage(Altem.Text + ' ' + Altem.ButtonText + ' foi clicado.');
```

```
end;
```

O que você pode entender desse código é que o evento de clicar no botão tem como parâmetro o item e o objeto. Desse modo é possível saber o que foi clicado, qual índice, etc.

## Resultado

Em seguida, você pode executar o aplicativo em seu dispositivo móvel, seja pressionando F9 ou escolhendo *Run > Run*.



|    |                 |        |
|----|-----------------|--------|
| 47 | Cornflowerblue  | 43,43  |
| 19 | Blue            | 60,6   |
| 22 | Powderblue      | 15,15  |
| 36 | Lightsteelblue  | 169,68 |
| 8  | Steelblue       | 4,04   |
| 12 | Mediumslateblue | 183,82 |
| 32 | Skyblue         | 178,77 |
| 9  | Slateblue       | 29,29  |
| 16 | Dodgerblue      | 157,56 |
|    | Darkblue        | 59,59  |
| 18 | Cadetblue       | 55,55  |
| 13 | Deepskyblue     | 87,87  |
| 23 | Blueviolet      | 189,88 |

**Figura 51 – Resultado final.**

**Nota:** para não carregar os dados previamente e iniciar o aplicativo de maneira “limpa”, vá com o botão direito do mouse no componente *ListView* e marque a opção *Clear List*. Em seguida, no *FormShow* ou em outro evento qualquer, execute o código a seguir para ativar o *LiveBindings*.

```
procedure TForm1.FormShow(Sender: TObject);  
begin  
    LinkFillControlToField1.BindList.FillList;  
end;
```

Para limpar a lista o contrário é válido, como no código:

```
procedure TForm1.FechaLiveBindings;  
begin  
    LinkFillControlToField1.BindList.ClearList;  
end;
```

## 7. *ListBox*

---

Na plataforma móvel, o FireMonkey usa o componente *FMX.ListBox.TListBox* para apresentar uma visão de tabela. O desempenho pode ser lento no celular. Use o *TListView* se você quiser desenvolver aplicativos mais complexos, especialmente com grandes bases de dados.

Existem dois tipos de lista: a plana e agrupada. A lista agrupada só é compatível com o iOS, não possuindo suporte para o Android.

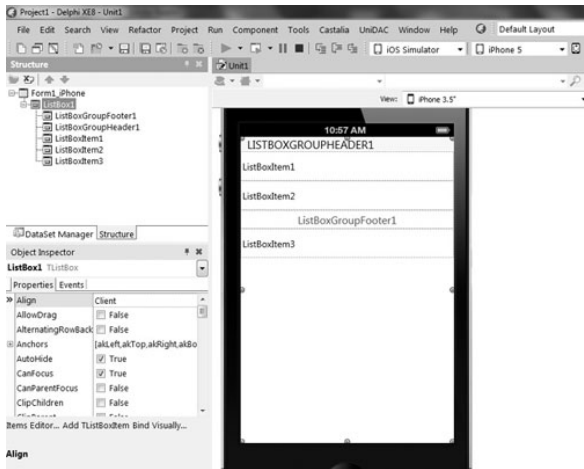
### Lista plana e lista agrupada

A diferença entre as listas é que, na lista agrupada, há uma quebra da lista e construção de uma nova lista, deixando um espaço vazio entre a próxima lista e dando a impressão de diversos componentes, sendo que é uma única lista, agrupada.

Já na lista plana, o funcionamento é o mesmo que um *TListView*, ou seja, plano.

Para criar um *ListBox* é muito simples: você deve criar um novo projeto ou executar um projeto já existente e, através do *ToolPalette*, buscar o *ListBox*.

1. Selecione o *ListBox* e altere as propriedades *Align* para *Client*, assim o componente ocupará todo o formulário.
2. Observe que é possível adicionar itens na lista estando em modo de desenho clicando com o botão direito do mouse na lista e selecionando, por exemplo, um *TSearchBox*, semelhante à caixa de pesquisa do *TListView*, porém, neste caso, no *ListBox*.



**Figura 52 – Grouped List no iPhone.**

## Adicionar um item ao *ListBox*

Para adicionar itens ao *ListBox* em modo de execução, basta executar o código:

```
ListBox1.Items.Add('Um item no listbox');
```

Ou

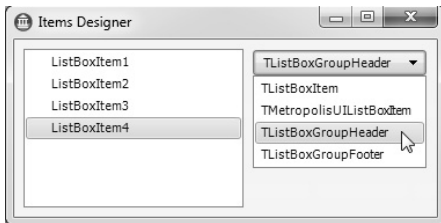
```
Var
ListBoxItem : TListBoxItem;
Begin
    ListBox1.BeginUpdate;
    ListBoxItem := TListBoxItem.Create(ListBox1);
    ListBoxItem.Text := 'Um item no listbox';
```

```

ListBox1.AddObject(ListBoxItem);
ListBox1.EndUpdate;
End;

```

Você pode adicionar um cabeçalho, tanto em modo de execução quanto em modo de desenho. Para isso basta selecionar o *ListBox* com o botão direito do mouse e adicionar um *TListBoxGroupHeader*. O mesmo vale para um possível rodapé, com o *TListBoxGroupFooter*.



**Figura 53 – Seleção de novo campo para o *ListBox*.**

Cada item em uma *TListBox* pode usar um acessório como uma marca de verificação por meio da propriedade *ItemData.Accessory*.



**Figura 54 – Valores para *ItemData.Accessory*.**

## 8. Acessando a Câmera

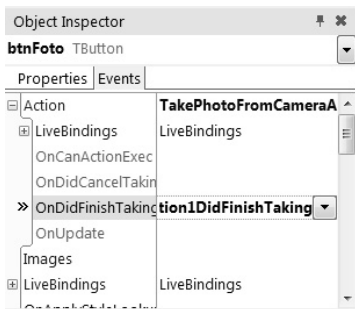
---

O FireMonkey suporta o mesmo conceito de *Actions* com o qual você, desenvolvedor Delphi<sup>\*</sup> mais antigo, já está acostumado.

Existem duas maneiras de interagir com as APIs de câmera, tanto para o Android, quanto para o iOS. A primeira que veremos aqui é a *TTakePhotoFromCameraAction*. O mais bacana disso tudo é que não é necessário codificar mais do que poucas linhas para interagir com a câmera, apenas utilizar a ação predefinida.

### ***TTakePhotoFromCameraAction*, a forma mais fácil**

1. Em um projeto, insira um botão qualquer em seu formulário.
2. Adicione um componente chamado *ActionList*.
3. Adicione um componente de imagem – por exemplo, um *TImage*. Se preferir, altere sua propriedade *Align* para *Client*, assim o efeito ficará mais legal.
4. Para deixar o visual do seu botão mais atraente e intuitivo, modifique a propriedade *StyleLookup* para algo semelhante a uma câmera.
5. Ainda focado em seu botão, perceba que a primeira propriedade, *Action*, agora pode ser alterada. Selecione *New Standard Action* e em seguida, em *Media Library*, selecione *TTakePhotoFromCameraAction*.
6. Na aba *Events*, ainda em seu botão, perceba que a palavra *Action* tem um sinal de (+), podendo ser expandido. Portanto, expanda-o e dê um duplo clique no evento *OnDidFinishTaking*, para programar o evento ao término da captura da imagem.



**Figura 55 – Evento *OnDidFinishTaking*.**

Você precisará programar essa linha de código, para copiar o conteúdo da câmera do seu *smartphone* para o componente *TImage*.

```
procedure TForm1.TakePhotoFromCameraAction1DidFinishTaking(Image:
TBitmap);
begin
    Image1.Bitmap.Assign(Image);
end;
```

**Nota: você pode habilitar o modo de edição para a foto recém-tirada. Basta ir na *Action* e habilitar a opção *Editable*.**



**Figura 56 – Opção *Editable*.**

Assim, é possível realizar o *crop* da sua foto, redimensionando-a.





**Figura 57 – Figura com modo de edição no Android.**

Assim sendo, quando o usuário do seu aplicativo tocar no botão, o Delphi® carregará a câmera em segundo plano e dará a opção de finalizar ou descartar. Ao finalizar a captura, a fotografia será copiada para o seu *TImage*, e, assim, você tem acesso à câmera do Android e do iOS da maneira mais rápida e fácil.

## ***TCameraComponent* – Princípios básicos**

Além de utilizar o aplicativo da câmera de um dispositivo físico Android e iOS, você pode programar o seu próprio aplicativo de câmera, através do componente *TCameraComponent*, nativo.

Com ele é possível utilizar o *flash*, por exemplo, para desenvolver uma lanterna, aplicativo presente com diversas funcionalidades nas lojas do Google e da Apple

que conseguem um bom número de *downloads* simplesmente por terem funcionalidades “legais” junto com a lanterna.

Outras características são o acesso à câmera frontal e traseira.

Porém, nesse caso, você precisará controlar uma série de fatores, como os eventos da aplicação, o tempo para apresentar a imagem, conforme o usuário mexe com o dispositivo de um lado para outro, fazendo assim a imagem mudar, entre outras coisas.

1. Primeiro, crie um novo projeto, *blank application*, e adicione o componente *TCameraComponent* ao seu formulário FireMonkey.
2. Para apresentar a imagem da câmera em tempo real, insira um componente do tipo *TImage*.
3. Insira inicialmente dois botões, um para iniciar a câmera e outro para parar a câmera.

É necessário controlar os eventos do seu aplicativo, como por exemplo se está ativo, se o usuário deixou em *background*, ou seja, em segundo plano, e até mesmo se o usuário finalizou o aplicativo. Para isso é necessário que você crie uma função para controlar esses eventos.

É necessário que você adicione ao *Uses* do seu projeto as seguintes *Units*: *FMX.Platform*, *FMX.Objects* e *FMX.Media*.

Declare no seu formulário a seguinte função:

```
function AppEvent(AAppEvent: TApplicationEvent; AContext: TObject): Boolean;
```

Agora implemente o seguinte código-fonte para controlar os eventos:

```
function TForm1.AppEvent(AAppEvent: TApplicationEvent;
  AContext: TObject): Boolean;
begin
  case AAppEvent of
    TApplicationEvent.WillBecomeInactive:
      CameraComponent1.Active := False;
    TApplicationEvent.EnteredBackground:
      CameraComponent1.Active := False;
```

```

TApplicationEvent.WillTerminate:
    CameraComponent1.Active := False;
end;
end;

```

Nesse código, é mapeado o estado do aplicativo para quando ele ficar inativo, em segundo plano, e for terminado, respectivamente.

Agora é necessário que você ative a chamada dessa função, através do método *Create* do seu formulário.

```

procedure TForm1.FormCreate(Sender: TObject);
var
    AppEventSvc: IFMXApplicationEventService;
begin
    if
        TPlatformServices.Current.SupportsPlatformService(IFMXApplicationEventService
        IInterface(AppEventSvc)) then
        AppEventSvc.SetApplicationEventHandler(AppEvent);
end;

```

Nesse método, é criada uma variável do tipo *IFMXApplicationEventService*, que está na unit *FMX.Platform*. Se o suporte for bem-sucedido, esse evento adicionará uma chave para monitorar o estado do aplicativo através da função anteriormente programada “AppEvent”. Portanto, qualquer alteração no estado do aplicativo fará com que a função seja executada.

**Nota: o componente de câmera também é suportado em plataformas Windows, ativando a webcam padrão.**

Para os botões, nomeie seus *captions* para “Ativar” e “Desativar” e insira o seguinte código, no evento *OnClick*:

```

procedure TForm1.btnStartCameraClick(Sender: TObject);
begin
    { ativar a câmera }

```

```
CameraComponent1.Active := True;
end;
```

```
procedure TForm1.btnStopCameraClick(Sender: TObject);
begin
    { desativar a câmera }
    CameraComponent1.Active := False;
end;
```

O próximo passo é controlar os *buffers* de entrada, ou seja, a própria imagem, e como esses *buffers* serão apresentados na tela do seu aplicativo. Por isso o componente *TImage*.

Antes é necessário criar uma *procedure* contendo a programação da transferência dos *buffers* da câmera para o componente *TImage* e depois programar a chamada dessa *procedure* toda vez que os *buffers* vierem diferentes, ou seja, todo o tempo.

Você pode criar o procedimento com o seguinte nome, por exemplo:

```
Procedure TForm1.PegarImagem;
```

E então o código para transferir os *buffers* da câmera para o componente de imagem:

```
Procedure TForm1.PegarImagem;
Begin
    CameraComponent1.SampleBufferToBitmap(img1.Bitmap, True);
End;
```

Agora você precisa executar o “PegarImagem” toda vez que os *buffers* estiverem disponíveis para serem transferidos.

O componente *TCameraComponent* só possui um único evento, o *OnSampleBufferReady*. Para ativá-lo, vá até a aba de eventos do componente e clique duas vezes no evento.

Codifique o seguinte no evento:

```

procedure TForm1.CameraComponent1SampleBufferReady(
  Sender: TObject; const ATime: Int64);
begin
  TThread.Synchronize(TThread.CurrentThread, GetImage);
end;

```

O código executa uma *thread* para sincronizar, em um processo paralelo à imagem vinda da câmera para o componente de imagem.

Pronto, este é o mínimo necessário para trabalhar com a câmera e criar o seu próprio aplicativo, sem necessidade de utilizar os recursos de *Actions*.

## ***TCameraComponent* – Utilizando o *flash***

Para utilizar o *flash* de uma câmera em um dispositivo Android ou iOS, não é necessário muito esforço: basta, primeiramente, executar todos os passos descritos no funcionamento do componente *TCameraComponent* e acrescentar o código:

```

if CameraComponent1.HasFlash then
  CameraComponent1.FlashMode := FMX.Media.TFlashMode.FlashOn;

```

O código mencionado verifica se o dispositivo possui *flash* – e, caso positivo, a propriedade *FlashMode* é configurada para *FlashOn*, sendo necessária a chamada através do *FMX.Media.TFlashMode.FlashOn*.

Para desligar o *flash* utilize *FlashOff*. Se preferir automático, utilize *FlashAuto*.

## ***TCameraComponent* – Câmera frontal e traseira**

De igual modo, utilizando o componente *TCameraComponent*, codifique o seguinte para trocar as câmeras:

```

{Câmera frontal}
CameraComponent1.Active := False;
CameraComponent1.Kind := FMX.Media.TCameraKind.FrontCamera;
CameraComponent1.Active := True;
{Câmera traseira}
CameraComponent1.Active := False;

```

```
CameraComponent1.Kind := FMX.Media.TCameraKind.BackCamera;  
CameraComponent1.Active := True;
```

Por padrão, o componente vem com a propriedade *Kind* configurada para *Default* – na maioria dos casos, para a câmera traseira.

## 9. Sensores

---

Hoje em dia, os aparelhos possuem vários sensores diferentes. Com eles é possível detectar a luminosidade de um local, a temperatura, a orientação, a posição através do GPS, entre outras coisas interessantes. Neste capítulo você aprenderá sobre cada um e como trabalhar com eles.

Inicie um novo projeto no Delphi criando um novo *blank application*.

1. Declare na **Uses** do seu projeto a seguinte *unit*: *System.Sensors*.
2. Crie no **Private** do seu formulário uma propriedade “FSensorAtivo”, do tipo *TCustomSensor*, e uma com nome “FShowInfo”, do tipo *Boolean*.

A classe *TCustomSensor* será a responsável por disponibilizar o suporte necessário aos sensores de cada plataforma, como, por exemplo, o acelerômetro.

Existem outros componentes, já devidamente encapsulados, como: *TLocationSensor*, *TMotionSensor* e *TOrientationSensor*. Estes são encontrados, através do *Tool Palette*, na seção *Sensors*.

### ***TCustomSensor* – Passos iniciais**

O *TCustomSensor* fornece propriedades que dão informações sobre o hardware real ou lógico (software) do sensor instalado em seu sistema, tais como nome, descrição, categoria, modelo ou fabricante.

Cada sensor que você pode usar é derivado da classe base chamada *TCustomSensor*.

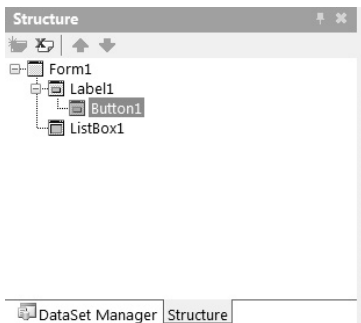
As propriedades *CustomData* e *CustomProperty* dão acesso aos valores dos sensores em propriedades específicas e campos de dados. As propriedades do sensor fornecem informações sobre as suas capacidades, como sensibilidade, fabricante, número do modelo e alcance. Um campo de dados do sensor dá os valores reais de medição que o sensor é projetado para determinar. Os campos de dados

são acessados apenas depois que as permissões de usuário são especificamente concedidas pelo usuário.

Quando o sensor envia dados, o evento *OnDataChanged* é acionado, tendo o papel de um notificador genérico. Ele não fornece informações sobre as propriedades que realmente mudaram seus valores. Da mesma forma, *OnStateChanged* é acionado quando a propriedade *State* muda o seu valor.

Com o seu projeto iniciado, e os itens 1 e 2 concluídos, insira um *ListBox*, um *Label* e um botão em seu formulário para que você possa capturar os sensores disponíveis em seu dispositivo e listá-los.

Defina a propriedade *Align* do seu *ListBox* para *Client* e certifique-se de que seu botão estará contido dentro da área do *Label*. Sim, isso é possível, você pode utilizar o *Label* como um *container*. Insira também um componente *Timer* (fora do *Label*) e deixe a propriedade *Enabled* como *True*, por padrão. Alinhe o seu botão dentro do *Label* como *Top*.



**Figura 58 – Estrutura do Form.**

Primeiro você deve criar duas funções auxiliares para buscar o tipo de sensor e apresentar em sua lista. A primeira delas é a que vamos chamar de *GetSensorCategoryName*.



```

function TForm1.GetSensorCategoryName(ASensorCategory: TSensorCategory): string;
begin
    Result := 'Não definido';
    case ASensorCategory of
        TSensorCategory.Location: Result := 'Location';
        TSensorCategory.Environmental: Result := 'Environmental';
        TSensorCategory.Motion: Result := 'Motion';
        TSensorCategory.Orientation: Result := 'Orientation';
        TSensorCategory.Mechanical: Result := 'Mechanical';
        TSensorCategory.Electrical: Result := 'Electrical';
        TSensorCategory.Biometric: Result := 'Biometric';
        TSensorCategory.Light: Result := 'Light';
        TSensorCategory.Scanner: Result := 'Scanner';
    end;
end;

```

Os sensores listados respectivamente são: localização, ambiental, movimento, orientação, mecânico, elétrico, biométrico, luz e scanner.

**Nota: os sensores podem não estar presentes em todos os aparelhos. O iPhone 4, por exemplo, não possui o sensor biométrico.**

Inicialmente você irá preencher a lista apenas com cabeçalhos contendo a categoria dos sensores. No decorrer deste capítulo, você terá acesso a cada um dos demais tipos de sensores, por categoria.

Na área de constantes, declare as seguintes:

```

Const
    AllCat : TSensorCategories =
        [TSensorCategory.Location, TSensorCategory.Environmental,
        TSensorCategory.Motion,
        TSensorCategory.Orientation, TSensorCategory.Mechanical,
        TSensorCategory.Electrical,

```

```
TSensorCategory.Biometric, TSensorCategory.Light, TSensorCategory.Scanner];
```

```
cForm = '%s=' + sLineBreak + '%30s' + sLineBreak;
cFormS = '%s=' + sLineBreak + '%30s' + sLineBreak;
```

A segunda função será responsável por apresentar os dados, em formato texto, quando o usuário tocar no nome do sensor, através do *ListBox*. Chame-a de *GetFullInfo*.

```
function TForm1.GetFullInfo(ACategory, AType, AClass,
    AAvailableProperties: string): string;
begin
    Result := sLineBreak + 'Category:' + sLineBreak
        + ' ' + ACategory + sLineBreak + sLineBreak
        + 'Sensor type:' + sLineBreak
        + ' ' + AType + sLineBreak + sLineBreak
        + 'Base class:' + sLineBreak
        + ' ' + AClass + sLineBreak + sLineBreak
        + 'Available properties:' + sLineBreak
        + AAvailableProperties;
end;
```

Essa função recebe os parâmetros do tipo texto e os retorna concatenados, para serem exibidos na lista.

Crie três funções necessárias para formatar os valores vindos dos sensores para o tipo *String*:

```
function ToFormStr(AProp : string; AVal : Single): string;
function ToFormStrS(AProp : string; AVal : string): string;
function ToFormStrB(AProp : string; AVal : Boolean): string;
```

As funções receberão dois parâmetros cada e retornarão em *strings* já formatadas.

```
function TForm1.ToFormStr(AProp: string; AVal: Single): string;
```

```
begin
  Result := Format(cForm,[AProp,"", Aval]);
end;
```

```
function TfrmAboutSensors.ToFormStrS(AProp, Aval: string): string;
begin
  Result := Format(cFormS,[AProp,"", Aval]);
end;
```

```
function TfrmAboutSensors.ToFormStrB(AProp: string; Aval: Boolean): string;
begin
  if Aval then
    Result := ToFormStrS(AProp,'Verdadeiro')
  else
    Result := ToFormStrS(AProp, 'Falso');
end;
```

Vamos listar todos os sensores disponíveis. No *Create* do seu formulário codifique o seguinte:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  LSensorCat : TSensorCategory ; //Categoria dos sensores
  LHeader : TListBoxGroupHeader; //Objeto Header (cabeçalho) do seu
  ListBox
begin
  FSensorAtivo := nil; //Declarado no item 2)
  FShowInfo := False; //Declarado no Item 2)

  //Ativa os sensores
  TSensorManager.Current.Activate();
  for LSensorCat in AllCat do
  begin
    //Cria um cabeçalho para o ListBox
    LHeader := TListBoxGroupHeader.Create(Owner);
    LHeader.Parent := ListBox1;
```

```

    LHeader.Text := GetSensorCategoryName(LSensorCat);
    LHeader.Height := LHeader.Height * 2;
end;
end;

```

De fato, nenhum tipo de sensor por categoria será exibido; porém, você já tem em seu formulário a sua lista categorizada por tipo de sensor.

## Sensor de luminosidade

Os sensores, embora sejam divididos em categorias, possuem outros subtipos ou tipos, como por exemplo o de luminosidade, que vem com o tipo *AmbientLight*. Para ter acesso a este sensor, você deve utilizar a propriedade *SensorType*.

Você pode criar uma função para retornar ao texto do item de sua *ListBox* o nome específico do tipo do seu sensor, categorizado.

```

function TForm1.GetTypeNameLight(AType: TLightSensorType): string;
begin
    Result := 'Não definido';
    case AType of
        TLightSensorType.AmbientLight: Result := 'AmbientLight' ;
    end;
end;
end;

```

Assim, você tem um objeto *TLightSensorType* que possui um tipo, chamado *AmbientLight* – no nosso bom português, luz ambiente. Experimente apagar a luz do seu quarto, à noite, para ver como ele funciona.

Como você acha que o seu aparelho controla o brilho da tela? Acertou se pensou no sensor de luminosidade.

Crie outra função, que será incrementada ao longo deste capítulo, para buscar o tipo de sensor, já categorizado. Inicialmente você trará apenas o de luminosidade.

```

function TForm1.GetSensorType(ASensor: TCustomSensor): string;
begin
    Result := 'Não definido';

```

```

Case ASensor.Category of
    TSensorCategory.Light: Result :=
        GetTypeNameLight(TCustomLightSensor(ASensor).SensorType);
end;
end;

```

De volta ao evento *OnCreate* do seu formulário, você deve incrementá-lo com alguns códigos e, assim, trazer as configurações do seu sensor de luminosidade.

```

procedure TForm1.FormCreate(Sender: TObject);
var
    LSensorCat : TSensorCategory ; //Categoria dos sensores
    LHeader : TListBoxGroupHeader; //Objeto Header (cabeçalho) do seu
    ListBox
    LSensorArray : TSensorArray; //Vetor contendo todos os sensores
    LSensor : TCustomSensor; //Sensor específico.
    LItem : TListBoxItem; //Item do ListBox
begin
    FSensorAtivo := nil; //Declarado no item 2)
    FShowInfo := False; //Declarado no Item 2)

    //Ativa os sensores
    TSensorManager.Current.Activate();
    for LSensorCat in AllCat do
        begin
            //Cria um cabeçalho para o ListBox
            LSensorArray := TSensorManager.Current.GetSensorsByCategory(LSensorCat);
            LHeader := TListBoxGroupHeader.Create(Owner);
            LHeader.Parent := ListBox1;
            LHeader.Text := GetSensorCategoryName(LSensorCat);
            LHeader.Height := LHeader.Height * 2;
            for LSensor in LSensorArray do
                begin
                    LItem := TListBoxItem.Create(Owner);
                    LItem.Parent := ListBox1;

```

```

        LItem.Text := GetSensorType(LSensor);
        LItem.ItemData.Accessory :=
            TListBoxItemData.TAccessory.aDetail;
        LItem.Height := LItem.Height * 2;
        LItem.Font.Size := LItem.Font.Size * 2;
    end;
end;
end;

```

No código apresentado, foram incluídas as variáveis *TSensorArray*, que percorrem, na ordem, os tipos de sensores por categoria. Como mostra o trecho:

```

LSensorArray := TSensorManager.Current.GetSensorsByCategory(LSensorCat);

```

A partir daí, um *ForIn* é executado, e, para cada tipo de sensor de uma determinada categoria encontrada, um novo item no *ListBox* será criado com o nome do tipo selecionado através da função *GetSensorType* que você programou anteriormente.

Além de apresentar o nome e o tipo do sensor quando o seu usuário tocar no item da lista, agora é necessário que o comportamento seja alterado, exibindo assim as propriedades do sensor. Para isso crie uma *procedure* para sobrescrever o evento *OnItemClick*, da sua *ListBox*. Chamei a *procedure* de *ListBoxItemClick*, com o parâmetro *Sender*, do tipo *TObject*.

```

procedure TForm1.ListBoxItemClick(Sender: TObject);
begin
    if Sender is TListBoxItem then
        FSensorAtivo := TCustomSensor(TListBoxItem(Sender).Data);
        FShowInfo := True;
end;

```

É necessário que você vá, através do *Object Inspector*, no seu componente *ListBox* e, na aba de eventos, codifique o evento *OnItemClick*.

```

procedure TForm1.ListBox1ItemClick(const Sender: TCustomListBox;

```

```

const Item: TListBoxItem);
begin
    if Assigned(Item.OnClick) then
        Item.OnClick(Item);
end;

```

Agora volte ao *OnCreate* do seu formulário e insira esse trecho de código dentro do *ForIn* do *SensorArray*.

```

LItem.Data := LSensor;
LItem.OnClick := ListBoxItemClick;

```

Significa que o item criado da lista recebe, na propriedade *Data*, que é um *TObject*, um objeto do tipo *TCustomSensor*, que no final das contas herda de *TObject*. É feito também o mapeamento do *OnClick*.

Você tem quase tudo pronto para monitorar, em tempo real, o sensor de luminosidade. Falta apresentar os dados do sensor ao clicar no item da lista – afinal, até o momento a única coisa que o *OnClick* realiza é configurar o valor da variável *FSensorAtivo* para o objeto *Data*, do item do *ListBox*.

Portanto, crie uma função para buscar as informações do sensor. Aqui vou chamá-la de *GetInfoAboutLight*.

```

function TForm1.GetInfoAboutLight(ASensor: TCustomSensor): string;
var
    Is : TCustomLightSensor;
    LValues : string;
    LProp : TCustomLightSensor.TProperty;
begin
    LValues := "";
    Is := TCustomLightSensor(ASensor);
    for LProp in Is.AvailableProperties do
        begin
            case LProp of
                TCustomLightSensor.TProperty.Lux:
                    LValues := LValues + ToFormStr('Lux', Is.Lux);
            end;
        end;
    end;
end;

```

```

    TCustomLightSensor.TProperty.Temperature:
        LValues := LValues + ToFormStr('Temperature', ls.Temperature);
    TCustomLightSensor.TProperty.Chromacity:
        LValues := LValues + ToFormStr('Chromacity', ls.Chromacity);
    end;
end;
Result := GetFullInfo(
    GetSensorCategoryName(ASensor.Category), GetTypeNameLight(ls.SensorType),
    ls.ClassName, LValues);
end;

```

Na função, você pode observar que existe uma classe específica para listar as propriedades do sensor de luminosidade, que neste caso é o *TCustomLightSensor*. O *ForIn*, em *ls.AvailableProperties*, exibe **somente** as propriedades suportadas no dispositivo físico.

No final, a função *GetFullInfo* que você criou nos **passos iniciais** serve para apresentar os dados do sensor, em formato texto, concatenados, já previamente formatados com as funções *ToFormStr*.

Agora, o último passo para exibição das propriedades do seu sensor de luminosidade. Você deve programar o evento do *Timer*, o *Timer1Timer*.

Coloque seu *Label1* com a propriedade *Align* como *Client* e defina a propriedade *Visible* para *False*.

```

procedure TForm1.Timer1Timer(Sender: TObject);
var
    ResultText : string;
begin
    if Assigned(FSensorAtivo) then
    begin
        case FSensorAtivo.Category of
            TSensorCategory.Light: ResultText := GetIn-
                foAboutLight(FSensorAtivo);
        end;
        lInfo.Text := ResultText;
    end;
end;

```



```

end;
if not FShowInfo then
begin
    Label1.Visible := False;
    ListBox1.Visible := True;
end;
else
begin
    Label1.Visible := True;
    ListBox1.Visible := False;
end;
end;

```

Por fim, você deve codificar o evento *OnClick* do seu botão, a fim de retornar para o estado anterior.

```

procedure TForm1.btnHideClick(Sender: TObject);
begin
    FShowInfo := False;
end;

```

Pronto, agora você pode executar o seu aplicativo. Repare que somente a opção para o sensor de luminosidade está disponível, pois foi a única a ser programada.

## Sensor magnético

Você já percebeu que o seu aparelho pode ser uma bússola? Sim, isso é verdade! Os dispositivos mais modernos já vêm equipados de fábrica com o que chamamos de sensor magnético. Isso significa que você pode desenvolver sua própria bússola!

A classe no Delphi<sup>®</sup> que controla esse sensor é o *TOrientationSensorType*, derivada de *TCustomOrientationSensorType*.

Para trabalharmos com o sensor magnético, você irá utilizar como base o exemplo contido nos **passos iniciais** e também seguindo o mesmo raciocínio do sensor de luminosidade. Para isso, incremente com os seguintes códigos:

```

function TForm1.GetTypeNameOrientation(
    AType: TOrientationSensorType): string;
begin
    case AType of
        TOrientationSensorType.Compass1D: Result := 'Compass1D';
        TOrientationSensorType.Compass2D: Result := 'Compass2D';
        TOrientationSensorType.Compass3D: Result := 'Compass3D';
        TOrientationSensorType.Inclinometer1D: Result := 'Inclinometer1D';
        TOrientationSensorType.Inclinometer2D: Result := 'Inclinometer2D';
        TOrientationSensorType.Inclinometer3D: Result := 'Inclinometer3D';
        TOrientationSensorType.Distance1D: Result := 'Distance1D';
        TOrientationSensorType.Distance2D: Result := 'Distance2D';
        TOrientationSensorType.Distance3D: Result := 'Distance3D';
    else
        Result := 'Não definido';
    end;
end;

```

Na função *GetSensorType*, você deverá incrementar com o seguinte código:

```

function TForm1.GetSensorType(ASensor: TCustomSensor): string;
begin
    Result := 'Não definido';
    Case ASensor.Category of
        TSensorCategory.Orientation: Result :=
            GetTypeNameOrientation(TCustomOrientationSensor(ASensor).SensorType);
        TSensorCategory.Light: Result :=
            GetTypeNameLight(TCustomLightSensor(ASensor).SensorType);
    end;
end;

```

Na função *GetTypeNameOrientation*, você provavelmente percebeu que o sensor dispõe de propriedades como: compasso 1, 2 e 3 (bússola), “inclinômetro” 1, 2 e 3 e distância 1, 2 e 3. Essas propriedades, juntas, formam o sensor de orientação ou magnético, sendo o compasso (bússola) para determinar o norte e o de

inclinação apenas para determinar se você está “trocando as pernas”, no caso de estar um pouco alcoolizado, através dos eixos X, Y e Z.

**Nota: no Android e iOS, somente está disponível o 3D, tanto para compasso (bússola) quanto para inclinação.**

O sensor de orientação utiliza o método *DoStart* para iniciar a recepção de orientação e notificações e o método *DoStop* para parar a recepção de orientação e notificações.

A classe *TCustomOrientationSensor* oferece informações sobre os eixos X e Y, e o ângulo do eixo Z, expresso em graus, através das propriedades *TiltX*, *TiltY* e *TiltZ*.

*MagHeading*, *TrueHeading*, *CompMagHeading* e *CompTrueHeading* retornarão o rumo da bússola em relação ao norte magnético e verdadeiro, podendo estar compassada ou descompassada, tendo em vista que uma bússola é uma espécie de “compasso magnético”.

As propriedades *DistanceX*, *DistanceY* e *DistanceZ* são *getters* para a distância no eixo, expressa em metros.

Para receber as informações do sensor magnético, uma vez que já foi codificado o *get* para o nome do sensor, codifique o seguinte:

```
function TForm1.GetInfoAboutOrientation(
    ASensor: TCustomSensor): string;
var
    ls : TCustomOrientationSensor;
    LValues : string;
    LProp : TCustomOrientationSensor.TProperty;
begin
    LValues := "";
    ls := TCustomOrientationSensor(ASensor);
    if not ls.Started then
        ls.Start;
    for LProp in ls.AvailableProperties do
        begin
```

case LProp of

TCustomOrientationSensor.TProperty.TiltX:

LValues := LValues + ToFormStr('TiltX', Is.TiltX);

TCustomOrientationSensor.TProperty.TiltY:

LValues := LValues + ToFormStr('TiltY', Is.TiltY);

TCustomOrientationSensor.TProperty.TiltZ:

LValues := LValues + ToFormStr('TiltZ', Is.TiltZ);

TCustomOrientationSensor.TProperty.DistanceX:

LValues := LValues + ToFormStr('DistanceX', Is.DistanceX);

TCustomOrientationSensor.TProperty.DistanceY:

LValues := LValues + ToFormStr('DistanceY', Is.DistanceY);

TCustomOrientationSensor.TProperty.DistanceZ:

LValues := LValues + ToFormStr('DistanceZ', Is.DistanceZ);

TCustomOrientationSensor.TProperty.HeadingX:

LValues := LValues + ToFormStr('HeadingX', Is.HeadingX);

TCustomOrientationSensor.TProperty.HeadingY:

LValues := LValues + ToFormStr('HeadingY', Is.HeadingY);

TCustomOrientationSensor.TProperty.HeadingZ:

LValues := LValues + ToFormStr('HeadingZ', Is.HeadingZ);

TCustomOrientationSensor.TProperty.MagHeading:

LValues := LValues + ToFormStr('MagHeading',  
Is.MagHeading);

TCustomOrientationSensor.TProperty.TrueHeading:

LValues := LValues + ToFormStr('TrueHeading',  
Is.TrueHeading);

TCustomOrientationSensor.TProperty.CompMagHeading:

LValues := LValues + ToFormStr('CompMagHeading',  
Is.CompMagHeading);

TCustomOrientationSensor.TProperty.CompTrueHeading:

LValues := LValues + ToFormStr('CompTrueHeading',  
Is.CompTrueHeading);

end;

end;

Result := GetFullInfo(

GetSensorCategoryName(ASensor.Category),

```

        GetTypeNameOrientation(Is.SensorType),
        Is.ClassName,
        LValues
    );
end;

```

Por fim, para realizar a codificação, altere os eventos do *Timer*:

```

procedure TForm1.Timer1Timer(Sender: TObject);
var
    ResultText : string;
begin
    if Assigned(FSensorAtivo) then
        begin
            case FSensorAtivo.Category of
                TSensorCategory.Light: ResultText := GetIn-
                    foAboutLight(FSensorAtivo);
                TSensorCategory.Orientation: ResultText :=
                    GetInfoAboutOrientation(FSensorAtivo);
            end;
            Label1.Text := ResultText;
        end;

    if not FShowInfo then
        begin
            Label1.Visible := False;
            ListBox1.Visible := True;
        end
    else
        begin
            Label1.Visible := True;
            ListBox1.Visible := False;
        end;
    end;
end;

```

Salve, execute e veja o resultado.

## Sensor de movimento

Você já experimentou arremessar o seu celular? Já experimentou, quando ainda criança, jogando uma partida em seu videogame, simplesmente realizar o movimento de inclinar para a esquerda ou para a direita, fingindo que o controle do seu videogame era na verdade um volante de um automóvel? Se respondeu sim, e eu espero que somente para a segunda pergunta, então você já percebeu o acelerômetro em dispositivos digitais na sua vida.

Os sensores de movimento no Delphi<sup>®</sup> são instanciados através da classe *TMotionSensor*, derivada de *TCustomMotionSensor*.

Ela oferece informações sobre aceleração, ângulo, estado e velocidade do movimento de um dispositivo.

O Android oferece diferentes tipos de sensores de movimento. Cada tipo de sensor suporta um conjunto diferente de propriedades:

- O sensor *Accelerometer3D* fornece as propriedades *AccelerationX*, *AccelerationY* e *AccelerationZ*.
- O sensor *Gyrometer3D* fornece as propriedades *AngleAccelX*, *AngleAccelY* e *AngleAccelZ*.
- O sensor *GravityAccelerometer3D* fornece as propriedades *AccelerationX*, *AccelerationY* e *AccelerationZ*.
- O sensor *LinearAccelerometer3D* fornece as propriedades *AccelerationX*, *AccelerationY* e *AccelerationZ*.

O iOS oferece diferentes tipos de sensores de movimento. Cada tipo de sensor suporta um conjunto diferente de propriedades:

- O sensor *Accelerometer3D* fornece as propriedades *AccelerationX*, *AccelerationY* e *AccelerationZ*.
- O sensor *MotionDetector* fornece as propriedades *AccelerationX*, *AccelerationY*, *AccelerationZ*, *AngleAccelX*, *AngleAccelY* e *AngleAccelZ*.

*AccelerationX*, *AccelerationY* e *AccelerationZ* retornam a aceleração em gals<sup>1</sup> para os eixos X, Y, Z.

|                       | m/s <sup>2</sup> | Pés/s <sup>2</sup> | Aceleração da gravidade (g0) | Gal (cm/s <sup>2</sup> ) |
|-----------------------|------------------|--------------------|------------------------------|--------------------------|
| 1 cm/s <sup>2</sup> = | 0,01             | 0,0328084          | 0,00101972                   | 1                        |

*AngleAccelX*, *AngleAccelY* e *AngleAccelZ* retornam a aceleração angular, em graus por segundo ao quadrado ( $^{\circ}/s^2$ ), para os eixos X, Y e Z.

*Speed* determina a velocidade do dispositivo em metros por segundo (m/s).

*Motion* determina se o dispositivo está ou não em movimento.

Para receber informações do sensor de movimento, incremente o exemplo deste capítulo com os seguintes códigos:

```
function TForm1.GetTypeNameMotion(AType: TMotionSensorType): string;
begin
    case AType of
        TMotionSensorType.Accelerometer1D: Result := 'Accelerometer1D';
        TMotionSensorType.Accelerometer2D: Result := 'Accelerometer2D';
        TMotionSensorType.Accelerometer3D: Result := 'Accelerometer3D';
        TMotionSensorType.MotionDetector: Result := 'MotionDetector';
        TMotionSensorType.Gyrometer1D: Result := 'Gyrometer1D';
        TMotionSensorType.Gyrometer2D: Result := 'Gyrometer2D';
        TMotionSensorType.Gyrometer3D: Result := 'Gyrometer3D';
        TMotionSensorType.Speedometer: Result := 'Speedometer';
        TMotionSensorType.LinearAccelerometer3D: Result :=
            'LinearAccelerometer3D';
        TMotionSensorType.GravityAccelerometer3D: Result :=
            'GravityAccelerometer3D';
    else
        Result := 'Não definido';
    end;
end;
```

Na função *GetSensorType*, incremente a cláusula *case*, desta vez com o *GetNameMotion*:

```
TSensorCategory.Motion: Result :=
    GetTypeNameMotion(TCustomMotionSensor(ASensor).SensorType);
```

Faça a codificação para que sejam retornadas as informações de cada propriedade do sensor de movimento:

```

function TForm1.GetInfoAboutMotion(ASensor: TCustomSensor): string;
var
    Is : TCustomMotionSensor;
    LValues : string;
    LProp : TCustomMotionSensor.TProperty;
begin
    LValues := "";
    Is := TCustomMotionSensor(ASensor);
    if not Is.Started then
        Is.Start;
    for LProp in Is.AvailableProperties do
        begin
            case LProp of
                TCustomMotionSensor.TProperty.AccelerationX:
                    LValues := LValues + ToFormStr('AccelerationX',
                        Is.AccelerationX);
                TCustomMotionSensor.TProperty.AccelerationY:
                    LValues := LValues + ToFormStr('AccelerationY',
                        Is.AccelerationY);
                TCustomMotionSensor.TProperty.AccelerationZ:
                    LValues := LValues + ToFormStr('AccelerationZ',
                        Is.AccelerationZ);
                TCustomMotionSensor.TProperty.AngleAccelX:
                    LValues := LValues + ToFormStr('AngleAccelX', Is.AngleAccelX);
                TCustomMotionSensor.TProperty.AngleAccelY:
                    LValues := LValues + ToFormStr('AngleAccelY', Is.AngleAccelY);
                TCustomMotionSensor.TProperty.AngleAccelZ:
                    LValues := LValues + ToFormStr('AngleAccelZ', Is.AngleAccelZ);
                TCustomMotionSensor.TProperty.Motion:
                    LValues := LValues + ToFormStr('Motion', Is.Motion);
                TCustomMotionSensor.TProperty.Speed:
                    LValues := LValues + ToFormStr('Speed', Is.Speed);
            end;
        end;
    end;
    Result := GetFullInfo(

```



```

    GetSensorCategoryName(ASensor.Category),
    GetTypeNameMotion(Is.SensorType),
    Is.ClassName,
    LValues
);
end;

```

No evento *Timer1Timer*, executar a chamada para o *GetInfoAboutMotion*:

```

case FSensorAtivo.Category of
  TSensorCategory.Motion: ResultText := GetInfoAboutMo-
tion(FSensorAtivo);

```

Salve e execute o projeto. Pronto!

Para outros sensores, você pode checar o exemplo da própria Embarcadero, através do link:

[http://docwiki.embarcadero.com/CodeExamples/XE7/en/FMX.Mobile.SensorInfo\\_Sample\\_\(Delphi\)](http://docwiki.embarcadero.com/CodeExamples/XE7/en/FMX.Mobile.SensorInfo_Sample_(Delphi)).

## Sensor de localização

Sem dúvida, um dos melhores sensores e também uma das mais avançadas invenções do mundo moderno é o GPS (*Global Positioning System*). Você sabe como funciona um GPS?

Atualmente são 27 satélites, em órbita, sendo 24 ativos e três reservas, para que os GPSs funcionem corretamente. Em cada canto do globo terrestre há pelo menos quatro deles; dessa forma, sempre que você e alguém dos Estados Unidos ou do Japão estiverem utilizando o GPS, não terão problemas.

Esses satélites mapeiam a sua posição através de um receptor que está contido na grande maioria dos aparelhos com Android e iOS. Esses receptores mostram sua posição, hora e outros recursos que variam de aparelho para aparelho.

Diferentemente dos outros sensores, o componente que gerencia o GPS, no Delphi®, foi encapsulado de forma mais elegante pela Embarcadero e transformado no *TLocationSensor*.

O *TLocationSensor* dispara um evento *OnLocationChanged* quando o dispositivo detecta o movimento. Você pode ajustar a sensibilidade do *TLocationSensor* usando as propriedades de distância e precisão.

A propriedade *Optimize* oferece a possibilidade de ligar ou desligar a otimização do sensor de localização.

*Accuracy* representa o nível de precisão com que o sensor detecta o dispositivo, em relação ao ponto geográfico em que está efetivamente localizado.

*Distance* é a distância mínima através da qual o dispositivo tem de mover, a fim de fazer a localização do sensor mudar o dispositivo e retornar nova informação de localização. Se você definir a propriedade como “5”, por exemplo, o *TLocationSensor* disparará um evento *OnLocationChanged* quando você se mover “5 metros”.

**Dica: você deve especificar a menor precisão que funcione para a sua aplicação; quanto maior for a precisão, mais tempo e energia o sensor necessitará para determinar a localização. Os valores recomendados são *Distance* = 0; *Accuracy* = 0.**

*TLocationSensor* oferece a possibilidade de interagir com a lista das regiões no local atual com métodos *AddRegion*, *RemoveRegion* e *ClearRegions*.

A região é uma área geográfica delimitada por um ponto geográfico e um raio em torno desse ponto. Um sensor de localização pode monitorar um conjunto de regiões e notificar seu aplicativo quando o dispositivo rodando o aplicativo entra ou sai de qualquer uma dessas regiões.

A propriedade *Regions* contém a lista das regiões que o sensor de localização está acompanhando.

**Nota: *Regions* é suportada apenas no iOS.**

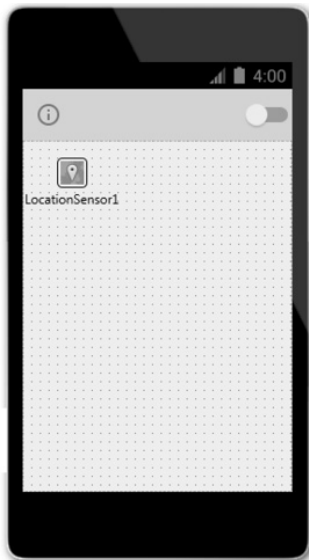
**Em dispositivos Android, o *TLocationSensor* requer permissões específicas para ser utilizado. Especificamente, as permissões *Access Coarse Location* e *Access Fine Location*.**

Para criar seu aplicativo com o sensor de localização, abra o Delphi® e crie um novo projeto, *blank application*, com os seguintes componentes:

1. Um *ToolBar* contendo dois componentes: um *TSwitch*, podendo ser alinhado à direita da tela, e um *TButton*, alinhado à esquerda.
2. Adicione um *TWebBrowser* alinhado como *Client*.
3. Por fim, adicione o componente *TLocationSensor*.



**Figura 59 – Aba Sensors.**



**Figura 60 – Formulário de exemplo.**

4. Selecione o componente na tela *TSwitch* e no *Object Inspector*. Na guia *Events*, clique duas vezes no evento *OnSwitch*.

```
procedure TForm1.Switch1Switch(Sender: TObject);  
begin  
    LocationSensor1.Active := Switch1.IsChecked;  
end;
```

5. Da mesma forma, selecione o *TLocationSensor* e, na guia de eventos, clique duas vezes no evento *OnLocationChange*.

```

procedure TForm1.LocationSensor1LocationChanged(Sender: TObject;
    const OldLocation, NewLocation: TLocationCoord2D);
var
    LDecSeparator: String;
    URLString: String;
begin
    LDecSeparator := FormatSettings.DecimalSeparator;
    FormatSettings.DecimalSeparator := '.';
    URLString := Format(
        'https://maps.google.com/maps?q=%s,%s',
        [Format('%2.6f', [NewLocation.Latitude]), Format('%2.6f',
            [NewLocation.Longitude])]);
    WebBrowser1.Navigate(URLString);
end;

```

O código mencionado executa, através do componente *TWebBrowser*, uma URL do Google Maps, passando as coordenadas de latitude e longitude dentro do método *Format*, substituindo os “coringas” *%s* pelos valores formatados seguindo o padrão *%2.6f*, que representa dois dígitos antes da casa decimal e seis dígitos após a casa decimal, no formato *float*. É necessário modificar o separador decimal para “.” (ponto), pois os servidores do Google não aceitam a vírgula.

Após montar a URL com as coordenadas, a função *Navigate* é executada e assim a sua localização atual será apresentada através do Google Maps.

Para testar, instale seu aplicativo em seu Android ou iOS e ative o *switch*. Faltou apenas programarmos o evento do botão, mas para isso veremos um pouco sobre *Geocoder*.

## Geocoding e reverse geocoding

Geocodificação é o processo de transformação dos dados geográficos, como endereço e códigos postais, em coordenadas geográficas. Geocodificação reversa é o processo de transformação de coordenadas geográficas em outros dados geográficos, tais como endereço. O procedimento de inicialização deve ser utilizado antes de usar o *TGeocoder* pela primeira vez, através da propriedade *CurrentY*. A função

*Supported* determina se a codificação geográfica pode ser realizada. *Authorized* determina se o aplicativo está autorizado a usar o serviço.

*Geocode* executa a geocodificação e dispara o evento *OnGeocode* quando terminar. *GeocodeReverse* executa a geocodificação reversa e dispara o evento *OnGeocodeReverse* quando terminar.

**Nota: no Android você não pode cancelar os pedidos de dados.**

A classe *TGeocoder* não é um componente, portanto é necessário que você codifique um pouco mais para utilizá-la.

Crie no seu *private* a variável *FGeocoder*, do tipo *TGeocoder*, e uma variável em formato texto para receber o endereço. Escreva também uma *procedure* para manipular o resultado da geolocalização.

```
Type
  TForm1 = class(TForm)
  private
    { Private declarations }
    FGeocoder: TGeocoder;
    FEndereco: String;
    procedure OnGeocodeReverseEvent(const Address: TCivicAddress);
  public
    { Public declarations }
  end;
```

```
procedure TForm1.OnGeocodeReverseEvent(const Address: TCivicAddress);
begin
  FEndereco := Address.AdminArea + '' +
    Address.CountryCode + '' +
    Address.CountryName + '' +
    Address.FeatureName + '' +
    Address.Locality + '' +
    Address.PostalCode + '' +
    Address.SubAdminArea + '' +
    Address.SubLocality + '' +
```

```

Address.SubThoroughfare + '' +
Address.Thoroughfare;
end;

```

O endereço atual, representado pela classe *TCivicAddress*, contém as informações detalhadas da geolocalização, como código de área, código do país, nome do país, local, código postal ou CEP e muito mais.

Voltando ao evento *OnLocationChanged*, você deve instanciar e ativar a classe de geolocalização, ou melhor, o *TGeocoder*. Faça isso da seguinte forma:

```

procedure TForm1.LocationSensor1LocationChanged(Sender: TObject;
const OldLocation, NewLocation: TLocationCoord2D);
var
  LDecSeparator: String;
  URLString: String;
begin
  LDecSeparator := FormatSettings.DecimalSeparator;
  FormatSettings.DecimalSeparator := '.';
  URLString := Format(
    'https://maps.google.com/maps?q=%s,%s',
    [Format('%2.6f', [NewLocation.Latitude]), Format('%2.6f',
    [NewLocation.Longitude])]);
  WebBrowser1.Navigate(URLString);

  try
    if not Assigned(FGeocoder) then
      begin
        if Assigned(TGeocoder.Current) then
          FGeocoder := TGeocoder.Current.Create;
        if Assigned(FGeocoder) then
          FGeocoder.OnGeocodeReverse := OnGeocodeReverseEvent;
        end;

        if Assigned(FGeocoder) and not FGeocoder.Geocoding then
          FGeocoder.GeocodeReverse(NewLocation);
      end
    except

```

```

    FEndereco := 'Geocoder error';
end;

end;

```

Perceba que o código anterior, que exibe o endereço no Google Maps através do *WebBrowser*, permanece o mesmo. A mudança efetiva está no bloco de código que instancia o *Geocoder* e dispara o evento *GeocodeReverse*.

*TGeocoder.Current* fornece o tipo de classe que realmente implementa o serviço de geocodificação. O código em *TGeocoder.Current.Create* chama o construtor (criar) para o tipo especificado e salva-o no campo *FGeocoder*.

Você também precisa especificar um manipulador de eventos, que é disparado quando *TGeocoder* completa a geocodificação reversa, atribuindo *OnGeocodeReverseEvent* para *FGeocoder.OnGeocodeReverse*.

Para apresentar o endereço atual, recuperado através do *TGeocoder*, foi incluído um botão. Você pode codificar o seu botão da seguinte forma:

```

procedure TForm1.btnEnderecoClick(Sender: TObject);
begin
    ShowMessage(FEndereco);
end;

```

Salve o seu projeto e execute-o em seu Android ou iOS.

---

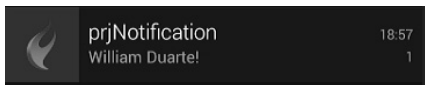
<sup>1</sup> Unidade de medida de aceleração da gravidade correspondente a  $1 \text{ cm/s}^2$ . Sua denominação é uma homenagem a Galileu Galilei.



# 10. Notificações

---

A maneira mais simples de entender uma notificação é olhar para o nosso dia a dia. Para os usuários mais atentos, é simplesmente o aviso quando seu amigo comenta uma publicação no Facebook ou quando você recebe uma nova mensagem, por exemplo, no WhatsApp.



**Figura 61 – Exemplo de notificação.**

Nas plataformas móveis, o FireMonkey fornece o componente *TNotificationCenter*, que permite acessar facilmente os serviços de notificação.

## Acesso ao serviço de notificações

Para trabalhar com o serviço de notificação, crie um novo projeto, podendo ser estilo *blank application*. Adicione as seguintes *units*:

```
uses  
FMX.Platform, FMX.Notification;
```

Selecione o componente *TNotificationCenter* na paleta de ferramentas (*Tool Palette*) e solte-o no formulário.

**Nota: se o componente *TNotificationCenter* estiver desativado, selecione a plataforma de destino para Android ou iOS.**

Para apresentar uma notificação imediatamente, você deverá utilizar o método *PresentNotification*.

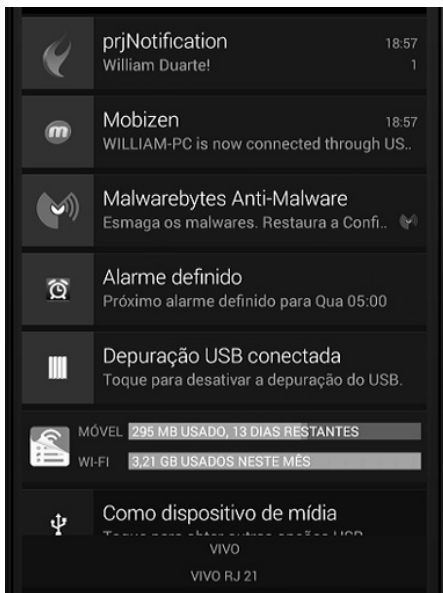
Crie dois botões em seu formulário e os renomeie. Por exemplo: “Disparar Notificação” e “Notificação Agendada”.

Primeiro codifique o botão “Disparar Notificação” da seguinte forma:

```
procedure TForm1.btnNotifcacaClick(Sender: TObject);
var
  MyNotification: TNotification;
begin
  MyNotification := NotificationCenter1.CreateNotification;
  try
    MyNotification.Name := 'LivroDelphi';
    MyNotification.AlertBody := 'William Duarte!';
    MyNotification.Number := 1;
    MyNotification.EnableSound := True;
    NotificationCenter1.PresentNotification(MyNotification);
  finally
    MyNotification.DisposeOf;
  end;
end;
```

No código mencionado, é criada uma variável do tipo *TNotification*, que caracteriza uma notificação. Essa notificação é criada dentro da central de notificações do dispositivo, por isso o *MyNotification := NotificationCenter1.CreateNotification*;

Toda notificação tem um nome ou um identificador – nesse caso, o identificador único dessa chamada é “LivroDelphi”. A mensagem que será apresentada é “William Duarte!”. As propriedades *Number* e *EnableSound*, respectivamente, caracterizam o número que será exibido, como, por exemplo, “Você tem 5 mensagens”, sendo o número 5 em destaque. A notificação pode ter som, configurado no dispositivo (o mesmo som que você recebe quando chega um novo e-mail, por exemplo).



**Figura 62 – Exemplo da notificação.**

## Apresentar uma notificação agendada

Você pôde aprender, no passo anterior, como apresentar uma notificação através da central de notificações do dispositivo utilizando o componente *TNotificationCenter* e a classe *TNotification*.

Para exibir uma notificação agendada não é preciso muita codificação, basta alterar alguns parâmetros e executar o método *ScheduleNotification*. Para tal, codifique o evento *OnClick* do botão “Notificação Agendada” da seguinte forma:

```

procedure TForm1.btnAgendadaClick(Sender: TObject);
var
    MyNotification: TNotification;
begin
    MyNotification := NotificationCenter1.CreateNotification;
    try
        MyNotification.Name := 'LivroDelphiAgendado';
        MyNotification.AlertBody := 'Notificação Agendada!';

        MyNotification.FireDate := Now + EncodeTime(0, 0, 10, 0);
        NotificationCenter1.ScheduleNotification(MyNotification);
    finally
        MyNotification.DisposeOf;
    end;
end;

```

As únicas diferenças em termos de codificação são:

1. *MyNotification.FireDate*, que recebe uma data/hora para agendamento – no caso, programei para 10 segundos à frente da hora exata.
2. *NotificationCenter1.ScheduleNotification(MyNotification)* envia para a central de notificações, como uma notificação agendada, sendo o serviço responsável por verificar se a data/hora chegou para que a notificação seja disparada.

## Atualizar ou cancelar uma notificação agendada

Cada notificação de mensagens agendadas é identificada através da propriedade *Name* do objeto *TNotification*.

Para atualizar uma notificação agendada, basta chamar o método *ScheduleNotification* novamente com uma instância de *TNotificationCenter* que tem o mesmo nome (propriedade *Name*).

Para cancelar uma notificação agendada, você pode simplesmente chamar o método *CancelNotification* com o identificador que você usou:

```

procedure TForm1.CancelNotification;

```

```
begin  
    NotificationCenter1.CancelNotification('LivroDelphiAgendado');  
end;
```

**Nota:** para a plataforma iOS, os métodos *CancelNotification* e *ScheduleNotification* podem cancelar ou atualizar somente as notificações que ainda não foram apresentadas. Se uma notificação já apareceu na lista de notificações do dispositivo, você não pode nem atualizar nem cancelar a notificação no iOS.

# 11. Permissões e Opções do Projeto

---

A definição de permissão de acesso para dispositivos móveis se caracteriza como o que o seu aplicativo pode ou não fazer em relação aos recursos de um dispositivo. Um exemplo disso é se o seu aplicativo pode ou não acessar a internet e ter acesso ao *wi-fi*. A forma de apresentação e concessão é diferente no Android e no iOS. De um lado, o Android exibe todas as permissões que determinado aplicativo necessita antes de executá-lo pela primeira vez. Já no iOS, a cada nova interação que necessita de uma permissão específica, é lançado na tela se o usuário permite ou não aquele comportamento.

A IDE do Delphi<sup>®</sup> é capaz de manipular essas permissões e, para os mais puristas, ela também é capaz de escrever nos arquivos base de configuração, como o *AndroidManifest.xml*, por exemplo.

Como no iOS as permissões são “por demanda”, não é necessária uma pré-configuração, diferentemente no Android, onde, através do *Project > Options > Uses Permissions*, é possível alterar os valores para verdadeiro ou falso, quando o seu aplicativo necessitar de alguma permissão específica.

Tenha um pouco de cuidado ao configurar as permissões, pois lojas de aplicativos como o Google Play poderão utilizar as permissões para filtrar as aplicações – por exemplo, se sua aplicação requer a permissão *call phone* (que já vem marcada por padrão no Delphi<sup>®</sup>), lojas de aplicativos podem não permitir que os usuários instalem o seu aplicativo em *tablets*, portanto é necessário um ajuste no arquivo de manifesto, o *AndroidManifest.xml*, para que o aplicativo funcione em *tablets* com a opção *call phone* habilitada. Basta adicionar a linha no manifesto:

```
<uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>
<uses-feature android:name="android.hardware.telephony" android:required="false"></uses-feature>
```

Alguns componentes FireMonkey exigem certas permissões. Por exemplo:

- *TCalendarEdit* requer *Read calendar* e *Write calendar*.
- *TLocationSensor* requer *Access coarse location* e *Access fine location*.

## Permissões padrão

Todo aplicativo FireMonkey para *mobile* já vem configurado com algumas permissões padrão para o Android. Caso o seu aplicativo não necessite de alguma permissão padrão, você tem a opção de modificá-la através do menu *Project > Options > Uses Permissions*.

As permissões padrão são essas:

- *Access coarse location*
- *Access fine location*
- *Call phone*
- *Camera*
- *Internet*
- *Read calendar*
- *Read external storage*
- *Write calendar*
- *Write external storage*
- *Read phone state*

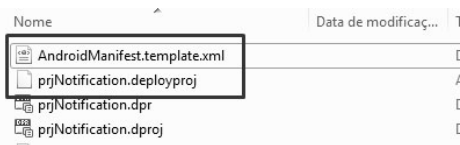
Você pode visualizar mais detalhes sobre cada permissão e o que ela “permite” através do link: [http://docwiki.embarcadero.com/RADStudio/XE7/en/Uses\\_Permissions](http://docwiki.embarcadero.com/RADStudio/XE7/en/Uses_Permissions).

## AndroidManifest.xml

Quando você cria um projeto para a plataforma de destino Android, o Delphi® gera um arquivo de manifesto do aplicativo Android. Este arquivo é necessário para gerar o conjunto de sua aplicação.

Quando você cria um aplicativo pela primeira vez, o Delphi® adiciona um arquivo para a pasta do projeto: *AndroidManifest.template.xml*. Sempre que você criar seu aplicativo para Android, o Delphi® lê este arquivo, substitui alguns espaços reservados no arquivo com valores reais de suas opções de projeto e escreve o conteúdo resultante em um arquivo de saída: *Android \ <build configuration> \ AndroidManifest.xml*. Este arquivo de saída está incluído no pacote do Android.

Para personalizar o arquivo *AndroidManifest.xml* para um único projeto, edite o conteúdo de *AndroidManifest.template.xml*.



**Figura 63 – Localização do *AndroidManifest.template.xml*.**

## Opções do projeto

Você deve ter percebido que, por padrão, todo aplicativo *mobile* gerado através do Delphi® vem com o ícone e a imagem *splash* com o logotipo do FireMonkey (uma insígnia de fogo).

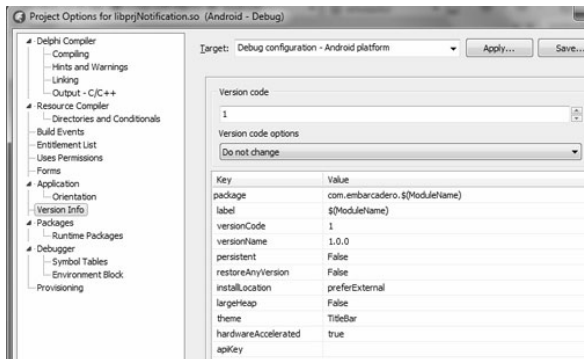
Além de modificar os ícones e imagens de *splash*, através do menu *Project > Options* é possível, como em versões anteriores, alterar os valores contendo o nome do projeto e sua versão. A novidade, porém, para as versões *mobile*, além dos diversos ícones, é a possibilidade de alterar o nome do pacote (sim, o pacote que as lojas virtuais de aplicativos utilizam) e até mesmo o local de instalação do aplicativo, seja na memória interna ou externa.

Duas opções bastante úteis dentro do *Project > Options* são:

- *Application*

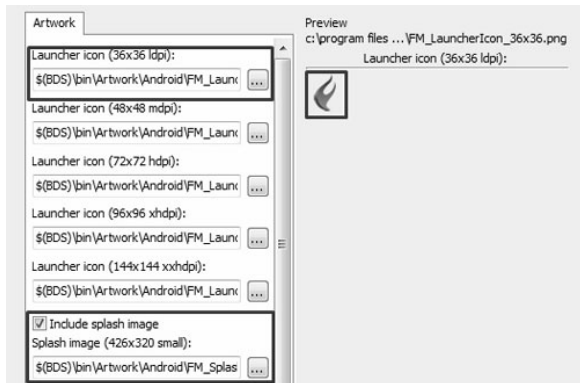


## • Version Info



**Figure 64 – Version Info.**

**Nota:** quando for enviar seu aplicativo para a loja virtual, lembre-se de trocar o valor do *package*, pois o padrão do Delphi® é “com.embarcadero.seuprojeto”. O ideal e por convenção é que você defina o nome de seu pacote assim: “br.com.suaempresa.seuapp”.



**Figura 65 – Application.**

Na página da aplicação, veja que é possível modificar o ícone do projeto para cada resolução específica. Isso é válido para que o seu aplicativo tenha a mesma iconografia, independentemente da resolução do dispositivo. E, é claro, para que o mesmo ícone apareça na loja virtual sem perda de qualidade. Observe também que é possível definir uma imagem de *splash*, ou seja, uma tela inicial enquanto o seu aplicativo carrega. Para isso, defina a opção *Include splash image* e defina o caminho das figuras, nas caixas de texto correspondentes, de acordo com o tamanho e a resolução.

Para maiores informações: [http://docwiki.embarcadero.com/RADStudio/XE7/en/Application\\_Options](http://docwiki.embarcadero.com/RADStudio/XE7/en/Application_Options).

## 12. API de Chamadas

---

O Delphi<sup>®</sup> vem com um conjunto de funções nativas que possibilitam o acesso à camada da API de chamadas, o *Phone Dialer Service*.

Com ele é possível não só realizar uma chamada telefônica, como recuperar o status de outras chamadas e até mesmo a informação da operadora.

Tudo o que você precisa é implementar a classe *TPhoneDialerService*.

### Primeiros passos: informações da operadora

Para o acesso à API, crie um novo projeto FireMonkey (pode ser *blank application*) e defina as seguintes *units* na sua cláusula *uses*:

```
FMX.PhoneDialer,  
FMX.Platform
```

No *private* do seu *Form*, declare a seguinte variável:

```
PhoneDialerService: IFMXPhoneDialerService;
```

Pronto, agora crie um *TListbox* em seu formulário e inclua nele quatro itens, que iremos chamar de:

1. Operadora
2. Código ISO do país
3. Código da rede
4. Código da operadora

Com a lista criada, inclua um botão para realizar a chamada à função. Agora você deve codificar seu botão da seguinte forma, a fim de trazer as informações da operadora:

```

procedure TPhoneDialerForm.btnGetCarrierInfoClick(Sender: TObject);
begin
    if TPlatformServices.Current.SupportsPlatformService(IFMXPhoneDialerService,
        IInterface(PhoneDialerService)) then
    begin
        CarrierNameItem.ItemData.Detail :=
            PhoneDialerService.GetCarrier.GetCarrierName;
        CountryCodeItem.ItemData.Detail :=
            PhoneDialerService.GetCarrier.GetIsoCountryCode;
        NetworkCodeItem.ItemData.Detail :=
            PhoneDialerService.GetCarrier.GetMobileCountryCode;
        MobileNetworkItem.ItemData.Detail :=
            PhoneDialerService.GetCarrier.GetMobileNetwork;
    end
    else
        ShowMessage('Serviço de Chamadas Não Suportado');
end;

```

No código, o trecho: *if TPlatformServices.Current.SupportsPlatformService(IFMXPhoneDialerService, IInterface(PhoneDialerService)) then* carrega para o objeto *PhoneDialerService* a biblioteca da API de chamadas. Caso o serviço seja suportado pelo aparelho ou se ele simplesmente possuir um *chip*, a propriedade *GetCarrier*, que retorna um objeto do tipo *TCarrier*, é então utilizada para que o *ListBox* seja preenchido.

A propriedade *GetCarrierName* retorna o nome da operadora – por exemplo, Vivo, Claro, Oi, Nextel.

A propriedade *GetIsoCountryCode* retorna o código ISO do país – por exemplo, BR.

A propriedade *GetMobileCountryCode* retorna o código do país para telefonia celular – por exemplo, no Brasil o código é 724.

A propriedade *GetMobileNetwork* retorna o código da operadora em conjunto com a faixa de MHz (megahertz) – por exemplo, 11 é da Vivo para as faixas GSM 850/GSM 1800/UMTS 850/UMTS 2100/LTE 2600.

**Nota:** antes de executar sua aplicação Delphi® em um dispositivo Android, verifique se as seguintes permissões estão definidas em *Project > Options > Uses Permissions*:

*Call phone*

*Read phone state*

## Realizando uma chamada

Para realizar uma chamada telefônica para algum número, a classe que você utilizará continuará a ser o *TPhoneDialerService*.

Inclua uma caixa de texto e um botão em seu formulário. Pode ser o mesmo formulário utilizado no passo inicial. Chame o botão de realizar chamada!

O código do botão para realizar uma chamada deverá ser semelhante ao seguinte:

```
procedure TPhoneDialerForm.btnMakeCallClick(Sender: TObject);
begin
  if TPlatformServices.Current.SupportsPlatformService(IFMXPhoneDialerService, IInterface(PhoneDialerService)) then
  begin
    if edtTelephoneNumber.Text <> '' then
      PhoneDialerService.Call(edtTelephoneNumber.Text)
    else
      begin
        ShowMessage('Insira um número.');
```

Novamente, a primeira linha verifica se o serviço está disponível e se a API de chamadas pode ou não ser executada. Em caso positivo, é verificado se a caixa de texto está vazia. Se não estiver, executa a função *Call(string)*.

Essa função, executada através do objeto *PhoneDialerService*, irá invocar a biblioteca interna do dispositivo e executará uma chamada telefônica para o número digitado. Muito simples, não!?

## Detectando o estado da chamada

Além de buscar informações da operadora e de realizar chamadas, é possível também verificar o seu estado atual. Como existem algumas diferenças de acordo com a plataforma de destino utilizada (Android ou iOS), segue uma tabela contendo o estado presente em cada uma. Tudo isso através da classe *TCallState*.

| Estado              | Descrição               | iOS | Android |
|---------------------|-------------------------|-----|---------|
| <i>None</i>         | Não há chamada em curso | Sim | Não     |
| <i>Connected</i>    | Chamada em curso        | Sim | Sim     |
| <i>Incoming</i>     | Recebendo uma chamada   | Sim | Sim     |
| <i>Dialing</i>      | Discando                | Sim | Não     |
| <i>Disconnected</i> | Chamada encerrada       | Sim | Sim     |

Para implementar o monitoramento de estados através da classe *TCallState*, você deve sobrescrever a chamada ao evento *OnCallStateChanged*.

Em seu formulário, podendo ser o mesmo desenvolvido ao longo deste capítulo, crie um *Label* para receber o texto contendo o estado da chamada e uma *procedure* na seção **private** do seu formulário principal, assim:

```
procedure MeuOnCallStateChanged(const ACallID: String; const ACallState: TCallState);
```

Criada a assinatura para o método, codifique sua implementação da seguinte forma:

```

procedure TForm1.MeuOnCallStateChanged(const ACallID: String; const
ACallState: TCallState);
var outText: String;
begin
case ACallState of
TCallState.None: outText := 'Sem chamadas';
TCallState.Connected: outText := 'Conectado';
TCallState.Incoming: outText := 'Recebendo ligacao';
TCallState.Dialing: outText := 'Discando';
TCallState.Disconnected: outText := 'Chamada encerrada';
end;
lblMeuLabel.Text := outText;
end;

```

Eu chamei o *Label* de “lblMeuLabel”, que receberá o texto de acordo com a máquina de estados toda vez que o estado atual da chamada for alterado.

Por fim, no *OnCreate* do seu formulário, modifique o *OnCallStateChanged* para o evento que você criou. Aqui neste exemplo, o *MeuOnCallStateChanged*.

```

TPlatformServices.Current.SupportsPlatformService(IFMXPhoneDialerService,
IInterface(PhoneDialerService));
if Assigned(PhoneDialerService) then
PhoneDialerService.OnCallStateChanged := MeuOnCallStateChanged;

```

## 13. FireDAC e SQLite

---

A Embarcadero, em uma jogada de mestre, presenteia a você e a todos os desenvolvedores Delphi® com um supercomponente para conexão e manipulação de banco de dados chamado FireDAC! Com ele atualmente é possível conectar-se a qualquer banco de dados e trabalhar, inclusive, com banco de dados embarcados, como por exemplo o SQLite, que na verdade é uma biblioteca em C que funciona como banco embarcado, já muito difundido em aplicações *mobile* por ser leve e de uso prático. Neste capítulo você verá como trabalhar com o FireDAC e como desenvolver suas aplicações utilizando o SQLite.

### Conceitos básicos

O FireDAC é um conjunto exclusivo de *Data Access Components* universais para o desenvolvimento de aplicações de banco de dados multiplataforma para Delphi® e C++ Builder®. Com sua poderosa arquitetura comum, FireDAC permite o acesso direto de alta velocidade nativa para o InterBase, SQLite, MySQL, SQL Server, Oracle, PostgreSQL, IBM DB2, SQL Anywhere, Access, Firebird, Informix e muito mais.

- Para as plataformas móveis, FireDAC suporta InterBase ToGo e SQLite. Esses produtos de banco de dados podem ser executados em dispositivos iOS e Android.
- Para outros bancos de dados, tais como Oracle, você precisa ter pelo menos uma biblioteca cliente. Em plataformas Windows, a biblioteca cliente é fornecida como uma DLL para se conectar. Portanto, você precisa desenvolver aplicativos usando tecnologias de camada intermediária como *DataSnap REST* para conectar-se a esses produtos de banco de dados a partir de um dispositivo móvel.



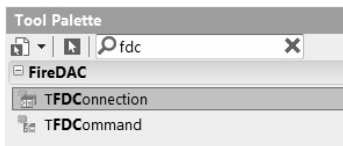
O SQLite é uma biblioteca em linguagem C que implementa um banco de dados SQL embutido. Programas que usam a biblioteca SQLite podem ter acesso a banco de dados SQL sem executar um processo SGBD separado.

Antes de iniciar o desenvolvimento *mobile*, você precisa criar o seu banco de dados SQLite no Windows, Mac ou Linux. Existem diversos softwares para criação e manipulação de bancos SQLite na internet. Neste capítulo utilizaremos o SQLiteStudio, que é gratuito. Você pode baixá-lo através do link: <http://sqlitestudio.pl/?act=download>

## Conexão com o FireDAC

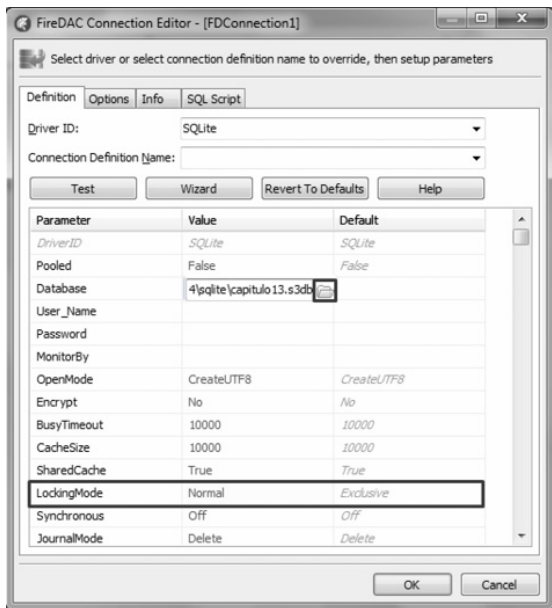
Crie um novo projeto (pode ser *blank application*). Para que você mantenha o código mais limpo, crie um *DataModule* através do menu *File > New > Other...* (em *Delphi Files*, selecione *Data Module*).

Com seu *Data Module* criado, procure pelo componente de conexão do FireDAC, o *TFDConnection*, e o insira na *data module*.



**Figura 66 – TFDConnection.**

Com o botão direito do mouse no componente *TFDConnection*, escolha “Connection Editor”.



**Figura 67 – Connection Editor.**

Defina os seguintes parâmetros:

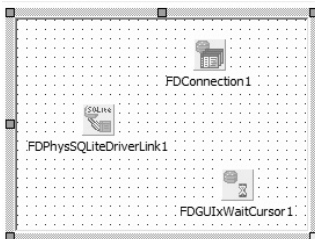
- Defina a propriedade do *DriverID* para *SQLite*.
- Defina o parâmetro de banco de dados para o arquivo que você salvou previamente em seu SQLiteStudio. No exemplo deste capítulo, chamei o banco de *capitulo13.s3db*.
- Defina a propriedade *LockingMode* para *Normal*.

**Nota: você pode definir o parâmetro de banco de dados, mesmo que *capitulo13.s3db* não exista (para exibir *capitulo13.s3db* ou outros arquivos *.s3db* na caixa de diálogo *Open*, defina todos os arquivos (\*.\*) – opção *All*).**

Clique em *Test*. Caso positivo, dê OK!

Existem outros componentes que você deve incluir no seu *Data Module* para que o Delphi® e o FireDAC não retornem erro ao realizar alguma operação no SQLite. Além do objeto de conexão, insira em seu *Data Module* os seguintes componentes:

- *FDPhysSQLiteDriverLink*
- *FDGUIxWaitCursor*



**Figura 68 – Driver e WaitCursor.**

O FireDAC possui uma arquitetura multicamadas fracamente acoplada, onde as camadas prestam serviços. Uma API de serviço é definida como uma interface COM onde outras camadas podem ser solicitadas através de uma fábrica de interfaces. Por esse motivo, para funcionar corretamente, você deve implementar os componentes de *Driver* e *Wait Cursor* na sua aplicação.

Para não mais exibir a mensagem de login com o banco de dados, selecione o componente de conexão e desmarque a opção *LoginPrompt* (configure para *False*).

Após configurar os parâmetros, configure a propriedade *Connected* para *True* (verdadeiro).

## BeforeConnect e AfterConnect

O objeto de conexão do FireDAC possui dois eventos bastante úteis: *BeforeConnect* (faça algo antes de conectar) e *AfterConnect* (faça algo após conectar).

É comum, em aplicações *mobile*, que o banco de dados não vá junto com o pacote de instalação. O procedimento mais indicado é, após efetuar corretamente a conexão com o banco de dados, verificar se determinada tabela existe. Se não existir, poderá criá-la. Com o seu projeto em andamento, programe o evento *BeforeConnect*.

```
procedure TDM_Conexao.FDConnection1BeforeConnect(Sender: TObject);
begin
    {$IF DEFINED(IOS) or DEFINED(ANDROID)}
    FDConnection1.Params.Values['Database'] :=
        TPath.GetDocumentsPath + PathDelim + 'capitulo13.s3db';
    {$ENDIF}
end;
```

O evento *BeforeConnect* é disparado no momento em que a propriedade *Connected* é transformada em *True* (verdadeiro), ou seja, antes de estabelecer a conexão, o bloco de código será executado.

A diretiva de compilação *{ \$IF DEFINED(IOS) or DEFINED(ANDROID) }* significa que determinado bloco de código **somente** será executado caso a plataforma de destino seja Android ou iOS.

Em seguida o objeto de conexão tem seu parâmetro *DataBase* modificado para o nome da pasta interna do dispositivo onde o aplicativo foi instalado, mais o nome do arquivo do banco de dados com a extensão “s3db”.

A comando *TPath.GetDocumentsPath* retorna o caminho onde o seu aplicativo está instalado no dispositivo físico. Para que essa linha seja executada é necessário que você defina na cláusula *Uses* do seu *Data Modulo* a seguinte *unit*: *System.IOUtils*.

Para que você avance um pouco mais, codifique o evento *AfterConnect* programando-o para criar determinada tabela, caso ela ainda não exista na base de dados. Você pode enviar um comando SQL diretamente através do componente de conexão usando o método *ExecSQL*.

```
procedure TDM_Conexao.FDConnection1AfterConnect(Sender: TObject);
begin
    FDConnection1.ExecSQL ('CREATE TABLE IF NOT EXISTS Item (ShopItem
        TEXT NOT NULL)');
end;
```

No código, disparado somente após a conexão com a base SQLite, o componente de conexão FireDAC envia, através do comando *ExecSQL*, o comando para criar, se não existir, a tabela “Item” contendo apenas uma coluna, chamada “ShopItem”, do tipo *TEXT*.

Você deve ter percebido que o SQLite, por não ser um banco de dados robusto, não possui diversos tipos – ou melhor, ele não tem todas as funcionalidades que um Oracle, por exemplo. Porém, para a realidade do mercado *mobile*, ele atende bem, muito bem.

A seguir, apresento uma tabela com os principais tipos e em seguida uma segunda tabela contendo a afinidade dos campos. Para maiores detalhes, visite a página do SQLite em: <https://www.sqlite.org/datatype3.html>.

| Tipo de dado   | Descrição  |
|----------------|--|
| <i>NULL</i>    | Valor nulo.  |
| <i>INTEGER</i> | Um valor como <i>integer signed</i> (com sinal) sem ser negativo. Guarda 1, 2, 3, 4, 6 ou 8 <i>bytes</i> , dependendo da magnitude do valor. |
| <i>REAL</i>    | Um valor como ponto flutuante. Guarda 8 <i>bytes</i> como <i>IEEE floating point number</i> .  |
| <i>TEXT</i>    | O valor é uma cadeia de texto, armazenado usando a codificação do banco de dados (UTF-8, UTF-16BE ou UTF-16LE).                              |
| <i>BLOB</i>    | Valor salvo <b>exatamente</b> como inserido na entrada.  |

**Nota: o SQLite não possui uma classe booleana separada de armazenamento. Em vez disso, os valores booleanos são armazenados como inteiros de 0 (falso) e 1 (verdadeiro).**

A fim de maximizar a compatibilidade com outros bancos de dados, o SQLite suporta o conceito de “afinidade de tipo” em colunas. A afinidade de tipo de uma coluna é recomendada para dados armazenados nessa coluna. A ideia importante aqui é que o tipo é recomendado, e não obrigatório. Qualquer coluna ainda pode armazenar qualquer tipo de dados. Algumas colunas, dada a escolha, vão preferir usar uma classe de armazenamento em detrimento de outra. A classe de armazenamento preferida para uma coluna é chamada de “afinidade”.

Existem cinco regras no SQLite para determinar a afinidade de uma coluna para outra:

1. Se o tipo declarado contém a *string* “INT”, ele é atribuído com a afinidade *INTEGER*.
2. Se o tipo declarado da coluna contém qualquer uma das *strings* “CHAR”, “CLOB”, ou “TEXT”, a coluna tem afinidade *TEXT*. Observe que o tipo *VARCHAR* contém a *string* “CHAR” e, portanto, é atribuído com a afinidade *TEXT*.
3. Se o tipo declarado da coluna contém a *string* “BLOB” ou se nenhum tipo for especificado, a coluna tem afinidade *NONE*.
4. Se o tipo declarado da coluna contém uma das *strings* “REAL”, “FLOAT”, ou “DUP”, a coluna tem afinidade *REAL*.
5. Caso contrário, a afinidade é *NUMERIC*.

| Exemplo de tipos vindos de um <i>CREATE TABLE</i> ou de uma expressão <i>CAST</i>                      | Afinidade      | Regra |
|--|----------------|-------|
| <i>INT</i><br><i>INTEGER</i><br><i>TINYINT</i><br><i>SMALLINT</i><br><i>MEDIUMINT</i><br><i>BIGINT</i> | <i>INTEGER</i> | 1     |

|   |         |   |
|---|---------|---|
| UNSIGNED BIG INT<br>INT2<br>INT8  |         |   |
| CHARACTER(20)<br>VARCHAR(255)<br>VARYING CHARACTER(255)<br>NCHAR(55)<br>NATIVE CHARACTER(70)<br>NVARCHAR(100)<br>TEXT<br>CLOB | TEXT    | 2 |
| BLOB ou nenhum tipo de dados especificado   | NONE    | 3 |
| REAL<br>DOUBLE<br>DOUBLE PRECISION<br>FLOAT   | REAL    | 4 |
| NUMERIC<br>DECIMAL(10,5)<br>BOOLEAN<br>DATE<br>DATETIME   | NUMERIC | 5 |

**Nota: observe que, além do tipo *BOOLEAN* ser um inteiro, o tipo *DATETIME*, por exemplo, não existe no SQLite, ele é guardado como um *NUMERIC*.**

Após essa pequena introdução ao SQLite, julgo que você esteja preparado para criar seus próprios bancos de dados e seguir em frente.

## FireDAC com *LiveBindings*

Antes de iniciar a configuração do *LiveBindings*, salve o seu *Data Module* e volte ao seu formulário principal. Nele, insira os seguintes componentes:

- *TToolBar*, sempre ao topo.

- *TButton*, dentro da *ToolBar*, com as propriedades de estilo modificadas para o formato ***addtoolbutton***. Mude o *caption* para “Inserir” e alinhe-o à direita.
- *TButton*, dentro da *ToolBar*, com as propriedades de estilo modificadas para o formato ***deletetoolbutton***, com o *caption* “Excluir”, alinhado à esquerda e com a propriedade *Visible* configurada para *False*.
- *Label*, também dentro da *ToolBar*, com as propriedades de estilo modificadas para o formato ***toollabel***, alinhado como *Client*, com o texto escrito “Lista de Compras” e com a propriedade *TextSettings* – precisamente o item *HorzAlign* configurado para *Center*.
- *ListView*, desta vez fora da *ToolBar*, com a propriedade de alinhamento configurada para *Client*.

Após configurar o seu formulário, salve o aplicativo e, se preferir, faça os ajustes que julgar necessários.

Inclua na cláusula *Uses* do seu formulário principal a *unit* que corresponde ao seu *Data Module*. No meu caso, a *unit* é “uDmConexao”.

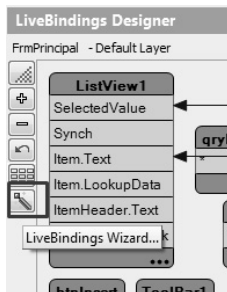




**Figura 69 – Layout proposto.**

Use o assistente de *LiveBindings* para adicionar os componentes *TBindSourceDB*, *TBindingsList* e *TFDQuery*.

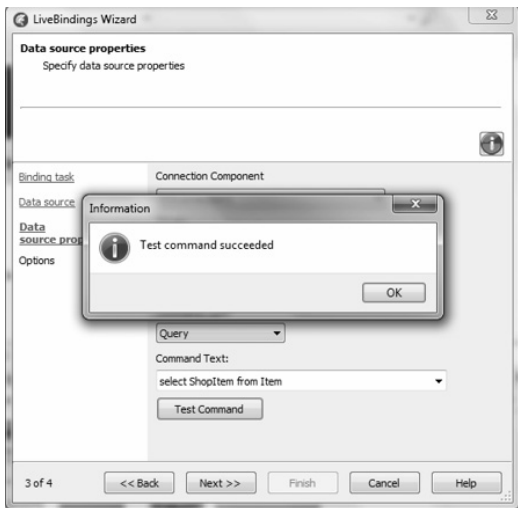
1. Selecione *View > LiveBindings Designer* e o *LiveBindings Designer* aparecerá abaixo da tela.
2. Selecione *LiveBindings Wizard*.



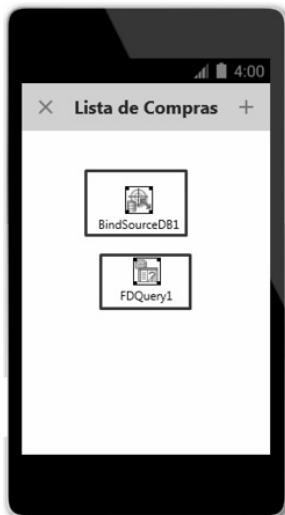
**Figura 70 – Clique no Wizard.**

3. Selecione “Create a DataSource” e clique em “Next”.
4. Selecione “FireDAC” e clique em “Next”.
5. Em “Connection Component”, se estiver vazio, inclua no seu formulário o *uses* onde está seu componente de conexão.
6. Em “Command Type”, mude para “Query”.
7. Em “Command Text”, digite: “select ShopItem from Item”.
8. Clique em “Test Command”.
9. Se tudo der certo, clique em “Next” e em seguida em “Finish”.

Neste ponto, os componentes *TBindSourceDB* e *TFDQuery* foram adicionados ao seu formulário.



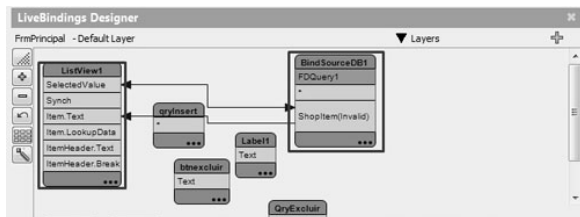
**Figura 71 – Teste OK.**



**Figura 72 – Formulário com componentes.**

O passo seguinte exibe o texto de *ShopItem* no componente *TListView*.

1. No *LiveBindings Designer*, selecione *ShopItem* no componente *BindSourceDB1* e arraste *ShopItem* para *Item.Text* em *ListView1*.



**Figura 73 – Exemplo.**

Em seguida ative a *Query*, com *Active* igual a *True*. Se você usou uma tabela com dados existentes, agora você deve ver os dados reais dentro do *Form Designer*.



**Figura 74 – Após ativar a Query.**

Perceba que, nesse exemplo, existem os produtos “Meu Teste” e “Produto”, pois foram inseridos através do SQLiteStudio.

A propriedade *Visible* para o botão *Excluir* está definida como *False*. Portanto, por padrão, o usuário final não vê esse botão. Você pode torná-lo visível quando o usuário selecionar um item na lista.

Para isso, codifique o evento *ItemClick* do seu *ListView*.

```
procedure TForm1.ListView1ItemClick(const Sender: TObject;  
const Altem: TListViewItem);  
begin  
    btnexcluir.Visible := ListView1.Selected <> nil;  
end;
```

O código mencionado somente deixará o botão de excluir visível quando um item selecionado for diferente de vazio.

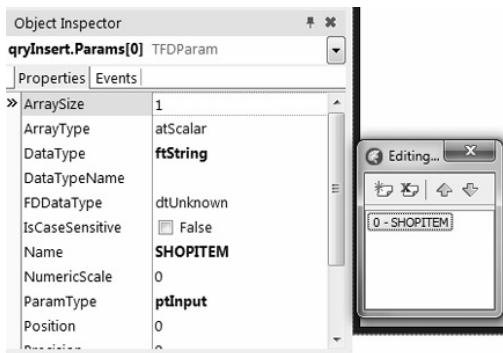
Agora, com metade do projeto realizado, faltam as funções para incluir um novo registro no banco (e, conseqüentemente, na lista) e para excluir um registro existente.

Para incluir um novo registro, você pode procurar o componente *TFDQuery*, que é o componente padrão para *Query* do FireDAC. Inclua um *TFDQuery* no seu formulário e configure a propriedade SQL como mencionado:

```
INSERT INTO ITEM (ShopItem) VALUES (:ShopItem)
```

Perceba que o comando SQL insere um registro na tabela *ITEM*, incluindo o valor do parâmetro (:ShopItem) dentro do campo *ShopItem*.

Em seguida, expanda a propriedade *Params* clicando em “(...)” e defina o parâmetro *ShopItem* com o *DataType* para *ftString*.



**Figura 75 – Parâmetros.**

**Nota: lembre-se de alterar a propriedade *Connection* do objeto *Query* para o objeto *FDConnection*, presente no seu *Data Module*.**

Lembrando que estamos em um ambiente *mobile* e que a preocupação com recursos e o custo de processamento deve ser levada em consideração. Você deve, ao menos nesse caso, como se trata de apenas um campo, utilizar os recursos disponíveis da linguagem para melhor lhe atender. O caso aqui é o *InputQuery*! Com o *InputQuery*, você pode escrever um texto em uma caixa de diálogo e consumir pouco recurso do dispositivo, pois ele não criará um novo formulário, por exemplo.

Para isso, declare a seguinte *procedure* na seção *private* do seu formulário.

```
private
procedure OnInputQuery_Close(const AResult: TModalResult; const AValues:
array of string);
```

Agora codifique sua implementação:

```

procedure TFrmPrincipal.OnInputQuery_Close(const AResult: TModalResult;
const AValues: array of string);
var
    NomeDoProduto: String;
begin
    NomeDoProduto := string.Empty;
    if AResult <> mrOk then
        Exit;
    NomeDoProduto := AValues[0];
    try
        if (NomeDoProduto.Trim <> "")
            then
                begin
                    qryInsert.ParamByName('ShopItem').AsString :=
                        NomeDoProduto;
                    qryInsert.ExecSQL();
                    FDQuery1.Close();
                    FDQuery1.Open;
                    btnexcluir.Visible := ListView1.Selected <> nil;
                end;
    except
        on e: Exception do
            begin
                ShowMessage(e.Message);
            end;
    end;
end;

```

Nesse código, é possível perceber que foi criada uma variável para receber o nome do produto digitado, variável esta que é preenchida através do parâmetro *AValues[0]*. Como *AValues[]* é um vetor, você precisa somente da primeira posição, ou seja, 0. Porém, antes é verificado se o usuário confirmou, clicando em OK, através do *ModalResult*.

A próxima validação é feita através da *string* dentro da variável “NomeDoProduto”. Caso ela esteja vazia, a função de gravação não é executada.



Caso esteja preenchida, então o parâmetro *ShopItem* é preenchido em modo de execução para o conteúdo da variável que na linha seguinte executa o comando SQL para inserção, através do *ExecSQL*.

Uma última tarefa é executada com os comandos para fechar e abrir (*Close* e *Open*) na *Query1*, que são responsáveis por abrir os registros no banco e, através do *LiveBindings*, preencher a lista. Se o item selecionado for diferente de vazio, então o botão de excluir torna-se visível.

Caso ocorra algum erro inesperado dentro do bloco *Try..Except*, uma exceção é lançada para o usuário através de um *ShowMessage*, contendo a mensagem de erro (*e.Message*).

Código entendido, agora falta executar o evento *OnClick* do botão de inserção:

```
procedure TFrmPrincipal.btnInsertClick(Sender: TObject);
var
    Values: array[0 .. 0] of String;
begin
    Values[0] := String.Empty;
    InputQuery('Entre com o nome', ['Nome'], Values,
        Self.OnInputQuery_Close);
end;
```

A variável do tipo *array* é utilizada, pois o *InputQuery* espera um tipo *array*. Como em seu caso a inserção é de apenas uma única coluna, o nome, então o *array* já é previamente criado com apenas uma posição. Em seguida, para evitar algum “lixo de memória”, o conteúdo é definido para vazio através do *String.Empty*. Por fim, o comando *InputQuery* é executado solicitando um nome para o produto. Veja que o *OnInputQuery\_Close* é definido no último parâmetro, para que seja executado ao fechar a janela do *input*.

Para excluir um registro, os passos são semelhantes. Você deve:

1. Inserir um componente *TFDQuery* em seu formulário.
2. Alterar a propriedade SQL, contendo o *script* para exclusão de um registro.

```
delete from Item where ShopItem = :ShopItem
```

3. Expandir a propriedade *Params* e definir o parâmetro *ShopItem*, da mesma forma como foi definido na *query* de inserção, ou seja, com o *DataType* para *ftString*.
4. Codificar o evento *OnClick* do botão *Excluir*.

```

procedure TFrmPrincipal.btnexcluirClick(Sender: TObject);
var
    Nome: String;
begin
    Nome := ListView1.Selected.Text;
    try
        qryExcluir.ParamByName('ShopItem').AsString := Nome;
        qryExcluir.ExecSQL();
        FDQuery1.Close;
        FDQuery1.Open;
        btnexcluir.Visible := ListView1.Selected <> nil;
    except
        on e: Exception do
            begin
                ShowMessage(e.Message);
            end;
        end;
    end;
end;

```

O mesmo ocorre no código do botão Incluir: em vez de executar a chamada ao *InputQuery*, como o botão só fica visível quando um item na lista está selecionado, a variável *nome* recebe o texto do item selecionado através da leitura da propriedade *Text* do *ListView1.Selected*.

A *Query* para excluir então tem seu parâmetro preenchido com o nome do item selecionado, que será excluído. Em seguida a lista é recarregada através do *Close* e *Open* da *Query* ligada à lista através do *LiveBindings*.

Caso ainda existam itens a serem excluídos, a verificação é feita para a propriedade *Visible* do botão. Em caso de erro dentro do bloco *Try..Except*, uma exceção é lançada para o usuário e uma caixa de mensagem é exibida contendo a mensagem de erro original que está dentro da variável de exceção *e.Message*.

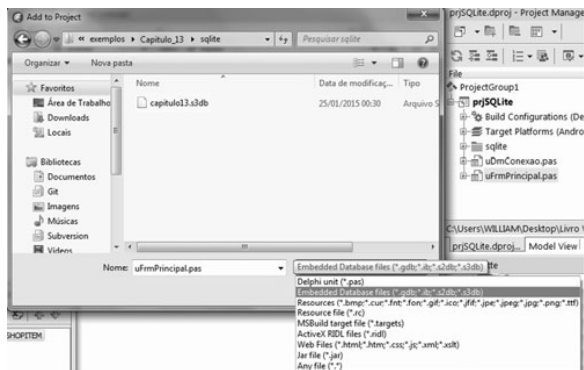
## Configurar seu banco de dados no dispositivo móvel

Até agora você deve ter usado o SQLite em seu *desktop*. Isso significa que o banco de dados real está localizado em sua unidade de disco rígido local. No dispositivo móvel, as aplicações são em *sandbox*, e normalmente você só pode ler e gravar dados que estão localizados na pasta Documentos (para dispositivos iOS) e de armazenamento interno (para dispositivos Android) em sua pasta de aplicativos.

Para se conectar a um banco de dados local no celular, você precisa executar as seguintes ações:

- Realizar o *Deploy*, através do *Deployment Manager*, do seu banco de dados no dispositivo móvel.
- Checar se o banco de dados existe e se as tabelas estão criadas (você viu isso nos eventos *BeforeConnect* e *AfterConnect*).

Você pode ir através do menu *Project > Add to Project* e selecionar seu arquivo de banco de dados.



**Figura 76 – Selecionando o banco.**

Depois de adicionar o arquivo de banco de dados, clique em “Cancelar” para fechar a janela que segue.

Novamente em *Project > Deployment*, observe que o banco de dados está adicionado aos arquivos de *Deploy* que serão carregados para o dispositivo quando seu aplicativo for instalado.

Veja como o caminho remoto (*remote path*) de seu arquivo (*.s3db*) foi definido para iOS e Android:

- iOS = StartUp\Documents\
- Android = .assets\internal\

Como você acabou de configurar, quando você executar o aplicativo no dispositivo móvel, o arquivo de banco de dados (*.s3db*) está previsto para ser implantado na pasta Documentos (para a plataforma iOS) ou armazenamento interno (para a plataforma Android) na área de *sandbox* de sua aplicação multidispositivo.

## Executando seu aplicativo

Antes de executar seu aplicativo e testar o SQLite em funcionamento, para melhor aproveitamento e correção de falhas que possam vir, crie um procedimento para que seu aplicativo saiba que entrou em estado de espera.

No ***private*** do seu formulário, declare o seguinte método:

```
procedure OnIdle(Sender: TObject; var FDone: Boolean);
```

E implemente da seguinte forma:

```
procedure TFrmPrincipal.OnIdle(Sender: TObject; var FDone: Boolean);
begin
    btnexcluir.Visible := ListView1.Selected <> nil;
end;
```

Codifique também o evento *OnCreate* do seu formulário para executar as chamadas ao banco de dados e ativar a conexão.

```
procedure TFrmPrincipal.FormCreate(Sender: TObject);
begin
```

```

try
    LinkFillControlToField1.AutoActivate := False;
    LinkFillControlToField1.AutoFill := False;
    Application.OnIdle := OnIdle;
    DM_Conexao.FDConnection1.Connected := True;
    FDQuery1.Active := True;
    LinkFillControlToField1.BindList.FillList;
except
    on e: Exception do
        begin
            ShowMessage(e.Message);
        end;
    end;
end;
end;

```

Perceba que as duas primeiras linhas se referem ao componente *BindingsList1*, que é o responsável por preencher a lista e também remover seus itens. Primeiramente a opção de *AutoActivate* (ativar automaticamente) é desativada, pois em *datasets* unidirecionais os conjuntos de dados são reativados toda vez que você utilizar o método *DataSet.First*. Por esse motivo, você não quer que toda vez que precise chamar o primeiro registro sua lista seja criada novamente.

O evento *Application.OnIdle* prevê que seu aplicativo pode entrar em espera, definindo que o botão ficará disponível apenas se houver um item selecionado na lista.

O *FDConnection1* é então executado para que a conexão seja feita com o banco SQLite, a *Query* que faz o *SELECT* em todos os produtos é ativada e, enfim, a lista é preenchida com as configurações do *LiveBindings*.

Antes de executar, troque a ordem de criação dos formulários, para que o *Data Module* seja criado **antes** do formulário, evitando assim que os componentes de *Query* não tenham a propriedade de conexão ativada.

```

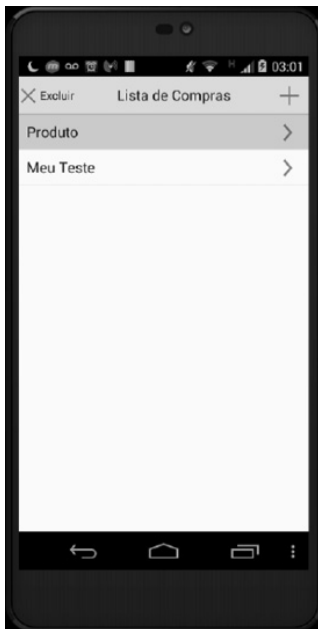
begin
    Application.Initialize;
    Application.CreateForm(TDM_Conexao, DM_Conexao);

```

```
Application.CreateForm(TFrmPrincipal, FrmPrincipal);  
Application.Run;  
end.
```

Inclua a linha de código no evento *OnClick* dos botões de incluir e excluir, para que a lista seja recarregada automaticamente.

```
LinkFillControlToField1.BindList.FillList;
```



---

**Figura 77 – Aplicativo em execução.**

# 14. Utilização de Linguagem Nativa

---

Existem aquelas situações em que você vai precisar de um recurso um pouco mais avançado, ou quem sabe uma característica mais particular que ainda não foi implementada no FireMonkey.

Para resolver esse impasse e trazer 100% de aproveitamento, o *framework* da Embarcadero permite que você importe classes do Java (para Android) e até mesmo Objective-C (para iOS).

**Nota: se você importar uma classe para Android, você perde a possibilidade de compilar o código para iOS e vice-versa. Uma forma de manter seus aplicativos multiplataforma é utilizar diretivas de compilação `{IFDEF}` para que seu compilador saiba que aquele código irá executar no Android e/ou no iOS.**

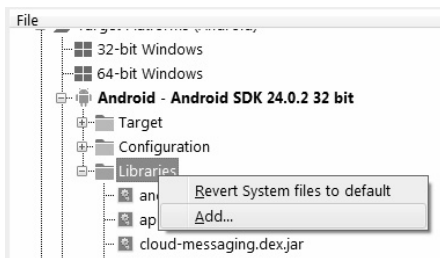
## Usando classes Java do SDK Android

Existem três formas de utilizar um código Java nativo em sua aplicação FireMonkey. A primeira delas é importar um arquivo (**.jar**), através do Delphi<sup>®</sup>, dentro do *Project Manager*. A segunda é utilizar JNI (*Java Native Interface*), através de mensagens *TOAST*, e a terceira, através de *intents*, quando você executa chamadas a outras APIs não só do Android como de qualquer outro aplicativo, como o Gmail e até mesmo o WhatsApp.

O Delphi<sup>®</sup> suporta apenas o nível Android API 10 (2.3.3 Android) e superior. Se as suas bibliotecas Java usam a API do Android, elas precisam estar na versão correta da API.



Para adicionar uma biblioteca Android, vá no *Project Manager* e, com a plataforma Android selecionada, procure a pasta *Libraries*.



**Figura 78 – Adicione um jar.**

Após adicionar o arquivo (.jar), você precisa criar um arquivo “ponte” entre seu aplicativo Delphi® e sua classe Java. Para isso é necessário criar arquivos *Bridges* nativos.

Você pode criar arquivos para Delphi® usando a ferramenta de linha de comando Java2OP.

Mas, afinal, o que é esse Java2OP (*Java to Object Pascal*)?

Java2OP.exe (“Java Para Object Pascal”) é uma ferramenta de linha de comando que você pode usar para gerar arquivos ponte nativos de bibliotecas JAR (Java) para Delphi®. Você pode usar os arquivos gerados para fornecer o seu acesso a aplicações Delphi® para essas bibliotecas Java no Android.

Java2OP.exe está disponível no site da Embarcadero somente para usuários registrados: <http://cc.embarcadero.com/myreg>.

Para maiores informações sobre o uso do Java2OP, veja o seguinte endereço:

[http://docwiki.embarcadero.com/RADStudio/XE7/en/Java2OP.exe,\\_the\\_Native\\_Bridge\\_File\\_Generator\\_for\\_Android](http://docwiki.embarcadero.com/RADStudio/XE7/en/Java2OP.exe,_the_Native_Bridge_File_Generator_for_Android).

Você pode utilizar as bibliotecas já importadas pela própria Embarcadero, que fornecem uma série de acessos já prontos, utilizando o conceito de JNI (*Java Native Interfaces*).

A instalação Delphi<sup>®</sup> inclui um conjunto de bibliotecas Java que são exigidas pelo FireMonkey como um todo ou por certos componentes e serviços de plataforma. O Delphi<sup>®</sup> também fornece uma ou mais **units** para cada uma dessas bibliotecas Java. Você pode incluir esses arquivos em seus aplicativos para acessar a API das bibliotecas Java subjacentes.

A instalação do Delphi<sup>®</sup> também inclui um arquivo *classes.dex*, que é o resultado da conversão desse conjunto de bibliotecas Java em um único arquivo executável *Dalvik*. Esse arquivo permite que seus aplicativos do Delphi<sup>®</sup> possam interagir com a API Android em modo de execução.

Por exemplo, é possível utilizar dentro do código-fonte uma biblioteca já pronta para o Android, que faz uma ponte com o código Java nativo.

A instalação do Delphi<sup>®</sup> inclui as seguintes bibliotecas Java:

| Biblioteca      | Unit   | Descrição  |
|-----------------|--|--|
| Android API     | Androidapi.JNI.App<br>Androidapi.JNI.Dalvik<br>Androidapi.JNI.GraphicsContentViewText<br>Androidapi.JNI.Hardware<br>Androidapi.JNI.InputMethodService<br>Androidapi.JNI.Java.Security<br>Androidapi.JNI.JavaTypes<br>Androidapi.JNI.Location<br>Androidapi.JNI.Media<br>Androidapi.JNI.Net<br>Androidapi.JNI.OpenGL<br>Androidapi.JNI.Os<br>Androidapi.JNI.Provider<br>Androidapi.JNI.Telephony<br>Androidapi.JNI.Util<br>Androidapi.JNI.VideoView<br>Androidapi.JNI.Webkit<br>Androidapi.JNI.Widget | API da plataforma Android  |
| Android Support | Androidapi.JNI.Support   | Conjunto de bibliotecas que oferece uma grande variedade de recursos |

| <b>Biblioteca</b>                 | <b>Unit</b>                     | <b>Descrição</b>  |
|-----------------------------------|---------------------------------|---|
|                                   |                                 | diferentes para a programação Android.  |
| FireMonkey                        | Androidapi.JNI.Embarcadero      | Biblioteca que fornece a funcionalidade principal exigida pelo FireMonkey Application Platform na plataforma Android. |
| Google Mobile Ads                 | Androidapi.JNI.AdMob            | SDK do serviço de anúncios para celular da Google.  |
| Google Analytics                  | Androidapi.JNI.Analytics        | SDK do serviço Google Analytics.  |
| Google Play Application Licensing | Androidapi.JNI.Licensing        | Uma coleção de classes auxiliares para simplificar a adição de licenciamento para uma aplicação Android.              |
| Google Play In-app Billing        | Androidapi.JNI.InAppBilling     | API do Google Play In-app Billing service.  |
| Google Play Services              | Androidapi.JNI.PlayServices     | API do Google Play Services.  |
| APK Expansion                     | Androidapi.JNI.ApkExpansion     | SDK para hospedagem de arquivos Android APK de expansão nos servidores Google Play.                                   |
| Cloud Messaging                   | Androidapi.JNI.PlayServices.GCM | SDK do serviço Google Cloud Messaging.  |

A biblioteca FireMonkey Java é uma biblioteca da Embarcadero. A APK Expansion e as bibliotecas da *Cloud Messaging* são bibliotecas de terceiros que incluem

modificações pela Embarcadero. O resto das bibliotecas Java listadas na tabela apresentada são bibliotecas de terceiros não modificadas.

## Exemplo – Atalho do aplicativo na área de trabalho

Um exemplo para trabalhar com algumas funções específicas do Android utilizando bibliotecas já presentes é, por exemplo, criar um atalho do seu aplicativo Android na área de trabalho do seu usuário.

Primeiro, você deve criar uma permissão específica dentro do seu arquivo *AndroidManifest.xml* – permissão esta que não é possível realizar dentro da IDE do Delphi<sup>®</sup>, pelo menos até o momento. Altere o arquivo de **Template**, para que você não precise alterar o arquivo sempre, e sim apenas uma única vez.

Inclua a seguinte linha no seu arquivo de manifesto:

```
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT"/>
```

Após modificar seu arquivo de *Template*, em seu formulário principal, altere o evento *OnCreate*, conforme descrito:

```
Uses
Androidapi.JNI.GraphicsContentViewText, FMX.Helpers.Android,
Androidapi.JNI.JavaTypes, FMX.Platform.Android, AndroidApi.JniBridge,
AndroidApi.Jni.App,
AndroidAPI.jni.OS;

{$IFDEF ANDROID}
var
ShortcutIntent: JIntent;
addIntent: JIntent;
wlconIdentifier : integer;
wlconResource : JIntent_ShortcutIconResource;
{$ENDIF}
begin
{$IFDEF ANDROID}
```

```

ShortcutIntent := TJIntent.JavaClass.init(SharedActivityContext,
SharedActivityContext.getClass);
ShortcutIntent.setAction(TJIntent.JavaClass.ACTION_MAIN);

addIntent := TJIntent.Create;

addIntent.putExtra(TJIntent.JavaClass.EXTRA_SHORTCUT_INTENT,
TJParcelable.Wrap((shortcutIntent as ILocalObject).GetObjectID));
// aqui é preciso dar CAST no Intent,
// uma vez que não é feito em Delphi® por padrão.
addIntent.putExtra(TJIntent.JavaClass.EXTRA_SHORTCUT_NAME, StringToJString(
ation.Title));
addIntent.setAction(StringToJString('com.android.launcher.action.INSTALL_SHO

// pegar o identificador do ícone
wlconIdentifier :=
SharedActivity.getResources.getIdentifier(StringToJString('ic_launcher'),
StringToJString('drawable'),
StringToJString('com.embarcadero.HeaderFooterApplication'));
// se o nome do app mudar você deve mudar o nome do projeto.

wlconResource :=
TJIntent_ShortcutIconResource.JavaClass.fromContext(SharedActivityContext,
wlconIdentifier);
// setar o ícone para o atalho
addIntent.putExtra(TJIntent.JavaClass.EXTRA_SHORTCUT_ICON_RESOURCE, TJPa
celable.Wrap((wlconResource as ILocalObject).GetObjectID));

SharedActivityContext.sendBroadcast(addIntent);

{$ENDIF}

end;

```

Perceba que na cláusula **Uses** do formulário foram incluídas diversas *units* para acesso às APIs internas do Java. Isso tudo dentro do Delphi!

## Toast Message – Android

Um *Toast* é uma exibição que contém uma pequena mensagem rápida para o usuário.

Quando uma mensagem é mostrada ao usuário, aparece como uma mensagem flutuante sobre a aplicação. A ideia é ser o mais discreto possível. Dois exemplos são o controle de volume e a breve mensagem dizendo que suas configurações foram salvas.

Como você viu anteriormente, o Delphi<sup>®</sup> permite que você declare uma classe “ponte” entre seu aplicativo Delphi<sup>®</sup> e uma biblioteca Java. Não existe uma implementação padrão da Embarcadero para o desenvolvimento com *Toasts*, mas é possível criar sua própria classe “ponte”. Todos os direitos da classe a seguir são do desenvolvedor Brian Long!

<http://blog.blong.com>.

```
unit Androidapi.JNI.Toast;

//Java bridge class imported by hand by Brian Long (http://blong.com)

interface

uses
  Androidapi.JNIBridge,
  Androidapi.JNI.JavaTypes,
  Androidapi.JNI.GraphicsContentViewText,
  Androidapi.Helpers; //Inclusão desta Unit para compatibilidade com
  Delphi® XE7

type
  TToastLength = (LongToast, ShortToast);

  JToast = interface;

  JToastClass = interface(JObjectClass)
    ['{69E2D233-B9D3-4F3E-B882-474C8E1D50E9}']
    {Property methods}
    function _GetLENGTH_LONG: Integer; cdecl;
```

```

function _GetLENGTH_SHORT: Integer; cdecl;
{Methods}
function init(context: JContext): JToast; cdecl; overload;
function makeText(context: JContext; text: JCharSequence; duration:
Integer): JToast; cdecl;
{Properties}
property LENGTH_LONG: Integer read _GetLENGTH_LONG;
property LENGTH_SHORT: Integer read _GetLENGTH_SHORT;
end;

[JavaSignature('android/widget/Toast')]
JToast = interface(JObject)
['{FD81CC32-BFBC-4838-8893-9DD01DE47B00}']
{Methods}
procedure cancel; cdecl;
function getDuration: Integer; cdecl;
function getGravity: Integer; cdecl;
function getHorizontalMargin: Single; cdecl;
function getVerticalMargin: Single; cdecl;

function getView: JView; cdecl;
function getXOffset: Integer; cdecl;
function getYOffset: Integer; cdecl;
procedure setDuration(value: Integer); cdecl;
procedure setGravity(gravity, xOffset, yOffset: Integer); cdecl;
procedure setMargin(horizontalMargin, verticalMargin: Single); cdecl;
procedure setText(s: JCharSequence); cdecl;
procedure setView(view: JView); cdecl;
procedure show; cdecl;
end;

TJToast = class(TJavaGenericImport<JToastClass, JToast>) end;

procedure Toast(const Msg: string; Duration: TToastLength = ShortToast);

implementation

uses
    FMX.Helpers.Android;

```

```

procedure Toast(const Msg: string; Duration: TToastLength);
var
    ToastLength: Integer;
begin
    if Duration = ShortToast then
        ToastLength := TJToast.JavaClass.LENGTH_SHORT
    else
        ToastLength := TJToast.JavaClass.LENGTH_LONG;
    CallInUiThread(procedure
    begin
        TJToast.JavaClass.makeText(SharedActivityContext,
            StrToJCharSequence(msg),
            ToastLength).show
    end);
end;
end.

```

Você pode copiar esta *unit* e salvar em seu disco para importação em seus projetos.

**Nota: todos os métodos devem ser declarados com a convenção *cdecl* para ser compatível com a convenção de chamada para o Java.**

## Acessando *wi-fi* via API nativa no Android

Veja um exemplo de como recuperar o estado da rede *wi-fi* e o estado da rede móvel (3G, 4G) utilizando linguagem nativa através das JNIs. Copie o conteúdo desse código e salve um novo arquivo.

```

unit utNetwork;

interface

function IsConnected: Boolean;

```



```
function IsWiFiConnected: Boolean;
```

```
function IsMobileConnected: Boolean;
```

```
implementation
```

```
uses
```

```
    System.SysUtils,  
    Androidapi.JNI.Bridge,  
    Androidapi.JNI.GraphicsContentViewText,  
    Androidapi.JNI.JavaTypes,  
    Androidapi.Helpers;
```

```
type
```

```
    JConnectivityManager = interface;  
    JNetworkInfo = interface;
```

```
    JNetworkInfoClass = interface(JObjectClass)  
    ['{E92E86E8-0BDE-4D5F-B44E-3148BD63A14C}']  
    end;
```

```
    [JavaSignature('android/net/NetworkInfo')]  
    JNetworkInfo = interface(JObject)  
    ['{6DF61A40-8D17-4E51-8EF2-32CDC81AC372}']  
    {Methods}  
    function isAvailable: Boolean; cdecl;  
    function isConnected: Boolean; cdecl;  
    function isConnectedOrConnecting: Boolean; cdecl;  
    end;
```

```
TJNetworkInfo = class(TJavaGenericImport<JNetworkInfoClass, JNetworkInfo>)
```

```
    JConnectivityManagerClass = interface(JObjectClass)  
    ['{E03A261F-59A4-4236-8CDF-0068FC6C5FA1}']  
    {Property methods}  
    function _GetType_WiFi: Integer; cdecl;  
    function _GetType_WiMax: Integer; cdecl;  
    function _GetType_Mobile: Integer; cdecl;
```

```

{Properties}
property TYPE_WIFI: Integer read _GetType_WIFI;
property TYPE_WIMAX: Integer read _GetType_WIMAX;
property TYPE_MOBILE: Integer read _GetType_MOBILE;
end;

[JavaSignature('android/net/ConnectivityManager')]
JConnectivityManager = interface(JObject)
['{1C4C1873-65AE-4722-8EEF-36BBF423C9C5}']
{Methods}
function getActiveNetworkInfo: JNetworkInfo; cdecl;
function getNetworkInfo(networkType: Integer): JNetworkInfo; cdecl;
end;
TJConnectivityManager = class(TJavaGenericImport<JConnectivityManagerC
JConnectivityManager>)
end;

```

```
function GetConnectivityManager: JConnectivityManager;
```

```
var
```

```
    ConnectivityServiceNative: JObject;
```

```
begin
```

```
    ConnectivityServiceNative :=
```

```
    SharedActivityContext.getSystemService(TJContext.JavaClass.CONNECTIVITY
```

```
    if not Assigned(ConnectivityServiceNative) then
```

```
        raise Exception.Create('Could not locate Connectivity Service');
```

```
    Result := TJConnectivityManager.Wrap(
```

```
        (ConnectivityServiceNative as ILocalObject).GetObjectID);
```

```
    if not Assigned(Result) then
```

```
        raise Exception.Create('Could not access Connectivity Manager');
```

```
end;
```

```
function IsConnected: Boolean;
```

```
var
```

```
    ConnectivityManager: JConnectivityManager;
```

```
    ActiveNetwork: JNetworkInfo;
```

```
begin
```

```
    ConnectivityManager := GetConnectivityManager;
```

```

ActiveNetwork := ConnectivityManager.getActiveNetworkInfo;
Result := Assigned(ActiveNetwork) and ActiveNetwork.isConnected;
end;

function IsWiFiConnected: Boolean;
var
    ConnectivityManager: JConnectivityManager;
    WiFiNetwork: JNetworkInfo;
begin
    ConnectivityManager := GetConnectivityManager;
    WiFiNetwork :=
        ConnectivityManager.getNetworkInfo(TJConnectivityManager.JavaClass.TYP
    Result := WiFiNetwork.isConnected;
end;

function IsMobileConnected: Boolean;
var
    ConnectivityManager: JConnectivityManager;
    MobileNetwork: JNetworkInfo;
begin
    ConnectivityManager := GetConnectivityManager;
    MobileNetwork :=
        ConnectivityManager.getNetworkInfo(TJConnectivityManager.JavaClass.TYP
    Result := MobileNetwork.isConnected;
end;

end.

```

A classe executa chamadas à própria API do Android através de *Intents* e busca, em funções programadas em Delphi<sup>®</sup>, se o *wi-fi* está conectado, se a internet móvel está conectada e se está conectado à internet. Muito útil para verificações de *downloads* e *uploads*.

Porém, você deve atentar para as permissões de acesso do aplicativo. Lembre-se de marcar o acesso do aplicativo ao *NETWORK\_STATE*; caso contrário, seu aplicativo pode lançar a exceção conforme figura:

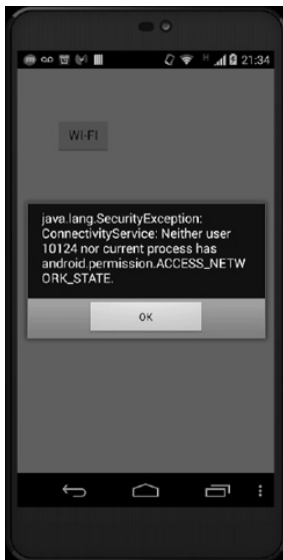


Figura 79 – Sem permissão.

## Usando classes Objective-C no iOS

Como você pôde ver na seção anterior, o Delphi<sup>®</sup> trabalha muito bem com linguagem nativa, e não é tão difícil implementar uma “ponte” para seus aplicativos. O mesmo ocorre com o iOS!

No Objective-C as classes têm um *NSObject* como ancestral comum, semelhante ao *TObject* do Delphi<sup>®</sup>.

Uma diferença que também você viu no Android é a parte de *strings*. No Delphi<sup>®</sup> temos *String* e no iOS temos *NSString* como objeto e não tipo primitivo. Para isso, da mesma forma que existe a classe *Helper* para o Android (Java),

existe também para o iOS, na *unit MacApi*.

*Helpers.pas*.

Você pode iniciar seu projeto incluindo a *unit iOSApi.Foundation* como base para suas chamadas Objective-C em Delphi<sup>®</sup>.

Para realizar a importação, atente para a tradução dos tipos *NSObjects* e para os *NSStrings*. Isso é muito importante: a conversão do que é chamado no Objective-C é transformado para o Delphi<sup>®</sup> através do tipo genérico:

```
TOCGenericimport<C: IObjectiveCClass; T: IObjectiveCInstance>
```

*TOCGenericimport* é uma classe genérica que podemos usar para tornar mais fácil a declaração de fábricas importadas para Objective-C. Usando essa classe, dividimos os métodos de classe e instância em duas interfaces. Essa classe combina as duas interfaces em uma única fábrica que pode produzir instâncias de objetos Objective-C ou fornecer uma referência a uma instância que representa a classe Objective-C.

Por exemplo, em Objective-C:

```
@property (nonatomic, retain) NSString * accessibilityLabel
```

No Delphi<sup>®</sup>:

```
function accessibilityLabel: NSString; cdecl;  
procedure setAccessibilityLabel(accessibilityLabel: NSString); cdecl;
```

## Acessando wi-fi via API nativa no iOS

Da mesma forma que foram apresentadas uma tradução e uma chamada às funções das bibliotecas Android nativas para o acesso ao escopo da rede, aqui também mostro um exemplo, em código-fonte, para acessar códigos Objective-C a fim de retornar o estado da rede.

```
unit utNetwork;  
  
interface
```

uses

Macapi.ObjectiveC, Macapi.CoreFoundation, Macapi.Dispatch,  
iOSSApi.CocoaTypes, iOSSApi.Foundation, Posix.SysSocket;

const

libSystemConfiguration =

‘/System/Library/Frameworks/SystemConfigura-  
tion.framework/SystemConfiguration’;

kSCNetworkFlagsConnectionAutomatic = 8;

kSCNetworkFlagsConnectionRequired = 4;

kSCNetworkFlagsInterventionRequired = 16;

kSCNetworkFlagsIsDirect = 131072;

kSCNetworkFlagsIsLocalAddress = 65536;

kSCNetworkFlagsReachable = 2;

kSCNetworkFlagsTransientConnection = 1;

kSCNetworkReachabilityFlagsConnectionAutomatic = 8;

kSCNetworkReachabilityFlagsConnectionOnDemand = 32;

kSCNetworkReachabilityFlagsConnectionOnTraffic = 8;

kSCNetworkReachabilityFlagsConnectionRequired = 4;

kSCNetworkReachabilityFlagsInterventionRequired = 16;

kSCNetworkReachabilityFlagsIsDirect = 131072;

kSCNetworkReachabilityFlagsIsLocalAddress = 65536;

kSCNetworkReachabilityFlagsReachable = 2;

kSCNetworkReachabilityFlagsTransientConnection = 1;

kSCNetworkReachabilityFlagsWWAN = \$40000;

Type

SCNetworkReachabilityFlags = UInt32;

SCNetworkReachabilityRef = ^\_\_SCNetworkReachability;

\_\_SCNetworkReachability = record

end;

SCNetworkReachabilityContext = record

version: CFIndex;

info: Pointer;

retain: function(info: Pointer): Pointer;

```

    release: procedure(info: Pointer);
    copyDescription: function(info: Pointer): CFStringRef;
end;
SCNetworkReachabilityContextPtr = ^SCNetworkReachabilityContext;
SCNetworkReachabilityCallback = procedure(target:
SCNetworkReachabilityRef;
    flags: SCNetworkReachabilityFlags; info: Pointer);
TReachability = class;

Reachability = interface(NSObject)
    ['B405394F-57B1-4FF1-83D9-8FBFA38FFD7B']
    function startNotifier: LongBool; cdecl;
    procedure stopNotifier; cdecl;
    function isReachable: LongBool; cdecl;
    function isReachableViaWWAN: LongBool; cdecl;
    function isReachableViaWiFi: LongBool; cdecl;
    function isConnectionRequired: LongBool; cdecl;
    function connectionRequired: LongBool; cdecl;
    function isConnectionOnDemand: LongBool; cdecl;
    function isInterventionRequired: LongBool; cdecl;
    function currentReachabilityStatus: NSInteger; cdecl;
    function reachabilityFlags: SCNetworkReachabilityFlags; cdecl;
    function currentReachabilityString: NSString; cdecl;
    function currentReachabilityFlags: NSString; cdecl;
end;
ReachabilityClass = interface(NSObjectClass)
    ['39EC0490-2787-4BB9-95EA-77BB885BFD01']
    function reachabilityWithHostname(hostname: NSString): Pointer;
    cdecl;
    function reachabilityForInternetConnection: Pointer; cdecl;
    function reachabilityWithAddress: Pointer; cdecl;
    function reachabilityForLocalWiFi: Pointer; cdecl;
end;
TReachability = class(TOCGenericImport<ReachabilityClass,
Reachability>)
end;

```

```

function SCNetworkReachabilityCreateWithAddress(allocator:
CFAllocatorRef;
    address: psockaddr): SCNetworkReachabilityRef; cdecl;
    external libSystemConfiguration name _PU +
    'SCNetworkReachabilityCreateWithAddress';
function SCNetworkReachabilityCreateWithAddressPair(allocator:
CFAllocatorRef;
    localAddress: psockaddr; remoteAddress: psockaddr):
    SCNetworkReachabilityRef;
    cdecl; external libSystemConfiguration name _PU +
    'SCNetworkReachabilityCreateWithAddressPair';
function SCNetworkReachabilityCreateWithName(allocator: CFAllocatorRef;
    nodename: PChar): SCNetworkReachabilityRef; cdecl;
    external libSystemConfiguration name _PU +
    'SCNetworkReachabilityCreateWithName';

function SCNetworkReachabilityGetTypeID: CTypeID; cdecl;
    external libSystemConfiguration name _PU +
    'SCNetworkReachabilityGetTypeID';
function SCNetworkReachabilityGetFlags(target: SCNetworkReachabilityRef;
    var flags: SCNetworkReachabilityFlags): Boolean; cdecl;
    external libSystemConfiguration name _PU +
    'SCNetworkReachabilityGetFlags';
function SCNetworkReachabilitySetCallback(target:
SCNetworkReachabilityRef;
    callout: SCNetworkReachabilityCallback;
    var context: SCNetworkReachabilityContext): Boolean; cdecl;
    external libSystemConfiguration name _PU +
    'SCNetworkReachabilitySetCallback';
function SCNetworkReachabilityScheduleWithRunLoop
    (target: SCNetworkReachabilityRef; runLoop: CFRRunLoopRef;
    runLoopMode: CFStringRef): Boolean; cdecl;
    external libSystemConfiguration name _PU +
    'SCNetworkReachabilityScheduleWithRunLoop';
function SCNetworkReachabilityUnscheduleFromRunLoop
    (target: SCNetworkReachabilityRef; runLoop: CFRRunLoopRef;

```



```

runLoopMode: CFStringRef): Boolean; cdecl;
external libSystemConfiguration name _PU +
'SCNetworkReachabilityUnscheduleFromRunLoop';
function SCNetworkReachabilitySetDispatchQueue(target:
SCNetworkReachabilityRef;
queue: dispatch_queue_t): Boolean; cdecl;
external libSystemConfiguration name _PU +
'SCNetworkReachabilitySetDispatchQueue';
{$IFDEF CPUARM}
function FakeLoader: Reachability; cdecl;
external 'libReachability.a' name 'OBJC_CLASS_$_Reachability';
{$ENDIF}

type
  TMobileNetworkStatus = class(TObject)
  public
    constructor Create;
    destructor Destroy; override;
    function isConnected: Boolean;
    function IsWiFiConnected: Boolean;
    function IsMobileConnected: Boolean;
  end;

implementation

function GetInternetReachability: Reachability;
begin
  Result := TReachability.Wrap
    (TReachability.OCClass.reachabilityForInternetConnection);
end;

constructor TMobileNetworkStatus.Create;
begin
end;

destructor TMobileNetworkStatus.Destroy;
begin
  inherited;

```

```

end;

function TMobileNetworkStatus.isConnected: Boolean;
begin
    Result := GetInternetReachability.isReachable;
end;

function TMobileNetworkStatus.IsMobileConnected: Boolean;
begin
    Result := GetInternetReachability.isReachableViaWWAN;
end;

function TMobileNetworkStatus.IsWiFiConnected: Boolean;
begin
    Result := GetInternetReachability.isReachableViaWiFi;
end;

initialization
{$IFDEF IOS}
{$IFDEF CPUARM}
if False then
    FakeLoader;
{$ENDIF}
{$ENDIF}
end.

```

**Nota: diferentemente do Android, no iOS não é necessário inserir as permissões antes, somente quando determinado aplicativo executar o recurso pela primeira vez.**

Para maiores informações, inclusive em vídeo, você pode checar o *Code Rage* de 2013, por Brian Long, que dá uma aula completa de como utilizar código nativo em suas aplicações. Veja no link: <http://blog.blong.com/2013/10/my-coderage-session-files.html>.

Para adicionar outros *frameworks* do seu iOS SDK, confira este endereço: <http://delphi.radsoft.com.au/2013/10/adding-other-ios-frameworks-to-the-sdk-manager/>.

Um exemplo de uso para importação de bibliotecas de terceiros está no blog do Fernando Rizzato. Veja no link: <http://blogs.embarcadero.com/fernandorizzato/index.php/2013/06/26/codigo-de-barras-no-delphi-para-ios/>.

Outra fonte de pesquisa, porém para OS X, está aqui: [http://delphiblog.twodesk.com/using\\_os\\_x\\_apis\\_directly\\_from\\_delphi](http://delphiblog.twodesk.com/using_os_x_apis_directly_from_delphi).

# 15. App Tethering

---

Introduzido na RTL (*Runtime Type Library*) desde a versão XE6, o conceito de *tether* (corrente, em inglês) provê a integração entre aplicativos, sejam *mobile* ou *desktop*. Não importa também se o seu aplicativo *desktop* foi desenvolvido com a VCL (*Visual Component Library*). O *app tethering* suporta isso!

É possível, através dessa “corrente”, ou seja, dessa integração, enviar uma fotografia recém-tirada em seu celular diretamente para seu *desktop*, ou quem sabe enviar um comando *True* ou *False*. As possibilidades são infinitas.

Trabalhando desde tipos primitivos como *String*, *Integer*, *Double* e *Boolean* até mesmo com *Streams*, é possível enviar e receber dados binários através de uma rede local ou através do *bluetooth*, facilitando e muito a sua vida como provedor de soluções.

Usando *app tethering*, seus aplicativos podem facilmente:

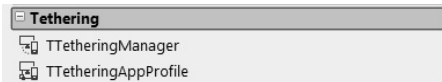
- Descobrir outros aplicativos que estão usando *app tethering*.
- Executar ações remotamente. Um aplicativo pode publicar ações usando *app tethering*. Em seguida, outras aplicações podem invocar remotamente qualquer uma dessas ações sobre a aplicação anterior.
- Compartilhar dados entre aplicações, tanto para tipos primitivos como para dados binários *streams*.

## Primeiros passos – Aplicativo servidor

O funcionamento do *app tethering*, como dito antes, está contido na RTL do Delphi<sup>®</sup> e também é capaz de funcionar em aplicações VCL, além do FireMonkey, o que é muito vantajoso! Crie um novo projeto, *blank application*, e inicialmente adicione os seguintes componentes:

- *TTetheringManager*

- *TTetheringAppProfile*



**Figura 80 – Aba Tethering.**

O componente *TTetheringManager* é utilizado para gerenciar e buscar os dispositivos que estão com o mesmo protocolo *tether*. Seja na mesma máquina ou em aparelhos diferentes, só depende do detalhe de estar **na mesma LAN** ou pareado via *bluetooth*.

O componente *TTetheringAppProfile* é utilizado para enviar e receber dados, ações e tudo o que for possível entre os dispositivos pareados através do *Manager*.

Nesse pequeno exemplo, você escreverá um servidor *desktop* para receber uma foto feita na hora, através de seu celular. Para isso, vá até o componente *Manager* e especifique as propriedades:

- **AllowedAdapters** – *Network*, para que o acesso seja feito via rede.
- **Password** – Defina uma senha qualquer. Será a mesma para o aplicativo *mobile*.
- **Enabled** – *True*.
- **Text** – Defina o texto para *ServidorTether*.

Para validar seu aplicativo cliente com o servidor, procure na aba de eventos o evento *OnRequestManagerPassword* e codifique o seguinte:

```
procedure TForm48.MediaReceiverManagerRequestManagerPassword(const
Sender: TObject; const RemoteIdentifier: string; var Password: string);
begin
    Password := '1234'; //Coloque a sua senha
end;
```

Vá até o componente *AppProfile* e altere as seguintes propriedades:

- **Manager** – Selecione no *combobox* o componente *Manager* já criado.
- **Enabled e Visible** – Deixe como *True*.

Antes que você programe os eventos de recebimento, adicione em seu formulário um componente de imagem (pode ser um *TImage*). Esse componente carregará na hora uma foto que você tirar com seu aplicativo *mobile*.

De volta ao componente *AppProfile*, vá até o evento *OnResourceReceived* e codifique a seguinte linha:

```
procedure TForm48.MediaReceiverProfileResourceReceived(const Sender:
TObject; const AResource: TRemoteResource);
begin
    Image1.Bitmap.LoadFromStream(AResource.Value.AsStream);
end;
```

Perceba que a constante, parâmetro da chamada, é carregada como *Stream* para dentro do componente *TImage*, pois sabe-se que o que virá é uma imagem mesmo. Fica ao critério do desenvolvedor, ou seja, seu, criar rotinas para validar e descobrir o tipo de dado enviado caso seu aplicativo permita o envio/recebimento de mais de um tipo (foto, texto, música, vídeo, etc).

Salve seu projeto e execute-o em seu Windows ou Mac, tanto faz (se estiver utilizando VCL, apenas no Windows).

## A mágica – Aplicativo cliente

Agora você vai desenvolver um aplicativo que executará a câmera do seu dispositivo e enviará uma fotografia recém-realizada, através da sua rede *wi-fi*, para seu aplicativo *desktop*, que estará aberto com um componente de imagem aguardando ansiosamente a foto.

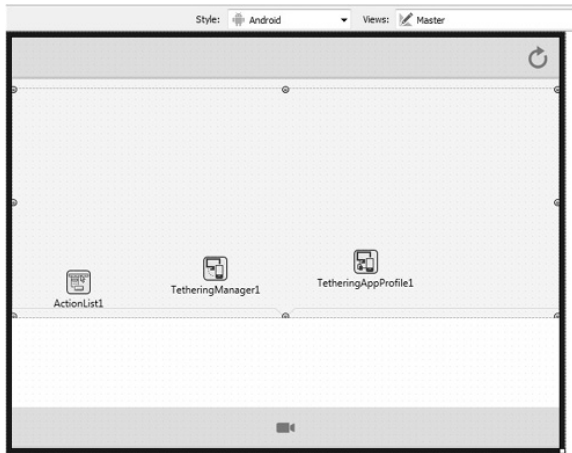
Crie um novo projeto *blank application* e adicione os mesmos componentes *Manager* e *AppProfile* para realizar o *tethering* entre seu aplicativo *mobile* e *desktop*.

Após criar os componentes *tethering* em seu aplicativo *mobile*, adicione outros:

- Dois *TToolBars*, sendo um alinhado ao topo e o segundo alinhado para baixo (*bottom*).
- No primeiro *toolbar*, adicione um botão, podendo modificar o seu estilo para *refreshtoolbutton*. Alinhe-o à direita.
- No segundo *toolbar*, adicione um botão, modificando seu estilo para *cameratoolbutton*. Alinhe-o ao centro.

- Adicione um *ListBox*, alinhe-o para baixo e certifique-se de que o *toolbar* esteja mais abaixo.
- Adicione um componente de imagem. Você pode inseri-lo dentro de um *TCallOutPanel*. Alinhe-o como *Client* (tanto o painel quanto a imagem).
- Adicione um *actionlist* para que você possa utilizar a ação padrão de executar a câmera.

Seu formulário deve ficar mais ou menos assim:



**Figura 81 – Exemplo de formulário.**

Chegou o momento de você programar os eventos e as propriedades dos componentes, a fim de criar seu aplicativo cliente. Primeiramente, crie uma variável no escopo *private* do seu formulário chamada ***Connected***, do tipo *boolean*.

```
private
{ Private declarations }
Connected: Boolean;
```

Em seguida, crie os seguintes procedimentos e funções:

```
procedure FindWalls;
function SendImage: Boolean;
function CheckPhotoWalls: Boolean;
procedure RefreshList;
```

- A *FindWalls* será para listar todos os servidores *tethering* pareados e desconectar um por um, além de limpar a lista.
- A *SendImage* será responsável por enviar o *stream* da imagem para o *Manager* do *desktop*.
- O *CheckPhotoWalls* serve como função apenas para validar se o envio para um servidor *tethering* está disponível.
- O *RefreshList* serve para atualizar sua lista e se conectar ao primeiro item.

Os códigos para cada procedimento/função são esses:

```
procedure TForm49.FindWalls;
var
    I: Integer;
begin
    LbWalls.Clear;
    for I := TakePhotoManager.PairedManagers.Count - 1 downto 0 do
        TakePhotoManager.UnPairManager(TakePhotoManager.PairedManagers[I]);
    TakePhotoManager.DiscoverManagers;
end;
```

```
function TForm49.SendImage: Boolean;
var
    LStream: TMemoryStream;
begin
    if not ImCaptured.Bitmap.IsEmpty then
    begin
        if not Connected then
            Connected :=
                TakePhotoAppProfile.Connect(TakePhotoManager.RemoteProfiles[LbWalls.Items[I].Text]);
    end;
end;
```



```

LStream := TMemoryStream.Create;
try
    ImCaptured.Bitmap.SaveToStream(LStream);
    Result :=
        TakePhotoAppProfile.SendStream(TakePhotoManager.RemoteProfiles
        'Foto do mobile', LStream);
finally
    LStream.Free;
end;
end
else
    ShowMessage('Fotografe algo antes de enviar');
end;

```

```

function TForm49.CheckPhotoWalls: Boolean;
begin
    if LbWalls.ItemIndex >= 0 then
        Result := True
    else
        begin
            Result := False;
            ShowMessage('Selecione um servidor para enviar');
        end;
    end;
end;

```

```

procedure TForm49.RefreshList;
var
    I: Integer;
begin
    LbWalls.Clear;
    for I := 0 to TakePhotoManager.RemoteProfiles.Count - 1 do
        if (TakePhotoManager.RemoteProfiles[I].ProfileText = 'ServidorTether')
            then
                LbWalls.Items.Add(TakePhotoManager.RemoteProfiles[I].ProfileText);
    if LbWalls.Count > 0 then

```

```

begin
    LbWalls.ItemIndex := 0;
    Connected :=
        TakePhotoAppProfile.Connect(TakePhotoManager.RemoteProfiles[0]);
end;
end;

```

Com todas as funções criadas, você deve configurar seus componentes *tethering*, os quais chamei aqui de *TakePhotoManager* para o *Manager* e *TakePhotoAppProfile* para o *AppProfile*. Atente para o *ServidorTether*, pois ele só irá trazer para a lista o perfil cujo texto seja este.

O nome do componente *ListBox* é *LbWalls*, enquanto a foto está como *ImCaptured*.

No componente *TakePhotoManager*, deixe o *AllowedAdapter* para *Network*, *Password* para *1234* e *Enabled* como *True*, exatamente como no aplicativo servidor!

No componente *TakePhotoAppProfile*, vincule a propriedade *Manager* e deixe-a ativada (*Enabled*, *True*).

Conforme você aprendeu no Capítulo 8, configure uma ação com *TakePhotoFromCameraAction*, através de seu *ActionList*, no botão na parte inferior do formulário.

Com a ação programada no botão, programe o seguinte código no evento *OnDidFinishTaking*:

```

procedure TForm49.TakePhotoFromCameraAction1DidFinishTaking(Image:
TBitmap);
begin
    ImCaptured.Bitmap.Assign(Image);
    if CheckPhotoWalls then
        SendImage;
end;

```

Já no botão de carregar, programe o seguinte:

```

procedure TForm49.BtRefreshClick(Sender: TObject);
begin
    FindWalls;
end;

```

Para o evento *OnShow* do formulário, faça o seguinte:

```

procedure TForm49.FormShow(Sender: TObject);
begin
    FindWalls;
end;

```

Caso você queira testar em mais de um computador ao mesmo tempo (por exemplo, você quer enviar para um segundo ou terceiro equipamento em sua empresa ou residência), com o mesmo aplicativo servidor instanciado, utilize o evento *OnItemClick* do seu *ListBox* para realizar a conexão com um servidor diferente:

```

procedure TForm49.LbWallsItemClick(const Sender: TCustomListBox; const Item: TListBoxItem);
begin
    if CheckPhotoWalls then
    begin
        TakePhotoAppProfile.Connect(TakePhotoManager.RemoteProfiles[LbWalls.ItemIndex]);
        LbWalls.Enabled := False;
        try
            SendImage;
        finally
            LbWalls.Enabled := True;
        end;
    end;
end;

```

Por fim, codifique os eventos do *Manager*, evento *RequestManagerPassword*:

```

procedure TForm49.TakePhotoManagerRequestManagerPassword(const
Sender: TObject; const RemoteIdentifier: string; var Password: string);
begin
    Password := '1234'; //sua senha
end;

```

Evento *RemoteManagerShutdown*:

```

procedure TForm49.TakePhotoManagerRemoteManagerShutdown(const
Sender: TObject; const ManagerIdentifier: string);
begin
    RefreshList;
end;

```

Evento *EndProfilesDiscovery*:

```

procedure TForm49.TakePhotoManagerEndProfilesDiscovery(const Sender:
TObject; const RemoteProfiles: TTetheringProfileInfoList);
begin
    RefreshList;
end;

```

Evento *EndManagersDiscovery*:

```

procedure TForm49.TakePhotoManagerEndManagersDiscovery(const
Sender: TObject; const RemoteManagers: TTetheringManagerInfoList);
var
    I: Integer;
begin
    for I := 0 to RemoteManagers.Count - 1 do
        if (RemoteManagers[I].ManagerText = 'ServidorTether') then
            TakePhotoManager.PairManager(RemoteManagers[I]);
end;

```

Salve seu aplicativo, execute-o em seu dispositivo, inicie o aplicativo servidor em seu *desktop* e experimente tirar uma foto do seu animal de estimação, por exemplo.

Para maiores detalhes e informações sobre aplicações com o conceito de *tethering*, visite:

- [http://docwiki.embarcadero.com/CodeExamples/XE7/en/RTL.DesktopCast\\_Sample](http://docwiki.embarcadero.com/CodeExamples/XE7/en/RTL.DesktopCast_Sample)
- [http://docwiki.embarcadero.com/RADStudio/XE7/en/Sharing\\_and\\_Running\\_Actions\\_on\\_Remote\\_Applications\\_Using\\_App\\_Tethering](http://docwiki.embarcadero.com/RADStudio/XE7/en/Sharing_and_Running_Actions_on_Remote_Applications_Using_App_Tethering)
- [http://docwiki.embarcadero.com/CodeExamples/XE7/en/RTL.PhotoWall\\_Sample](http://docwiki.embarcadero.com/CodeExamples/XE7/en/RTL.PhotoWall_Sample)
- [http://docwiki.embarcadero.com/CodeExamples/XE7/en/RTL.BDShoppingList\\_Sample](http://docwiki.embarcadero.com/CodeExamples/XE7/en/RTL.BDShoppingList_Sample)
- [http://docwiki.embarcadero.com/CodeExamples/XE7/en/RTL.MediaPlayer\\_Sample](http://docwiki.embarcadero.com/CodeExamples/XE7/en/RTL.MediaPlayer_Sample)

## 16. Media Player

---

Você já pensou em desenvolver seu próprio player de MP3? Ou quem sabe seu reprodutor de vídeo? E, se estiver com um pouco mais de paciência, um aplicativo juntando as funções do *media player* e do *tethering* seria muito legal, não é mesmo!?

O Delphi<sup>®</sup> vem com um componente nativo, multiplataforma, para que você realize todos os desejos descritos. Neste capítulo você aprenderá um pouco sobre esse componente e o que ele pode fazer.

### Aperte o play

Crie um novo projeto, pode ser *blank application*, e adicione o componente *TMediaPlayer*.

Para testar a utilização do seu componente multiplataforma, que tal embarcar um vídeo em formato MP4, através do *Deployment Manager* e executá-lo? Simples assim.

Crie um botão em seu formulário com o texto *Play* ou simplesmente mude o seu estilo para *Play*.

O código de seu botão poderá ficar assim:

```
MediaPlayer1.FileName :=  
IncludeTrailingPathDelimiter(TPath.GetDocumentsPath) +  
'SeuArquivo.mp4';  
MediaPlayer1.Play;
```

Para que isso funcione corretamente, declare em seu *uses* a seguinte *unit*:

Uses

```
System.IOUtils;
```

Logo abaixo crie um tipo, a fim de evitar erros na abertura do vídeo.

```
Type
  TOpenMedia = class(TMedia);
```

Pronto, seu aplicativo está feito e esperando que o usuário aperte o *play*!

## Aperte o rec

Com o componente *TMediaPlayer*, é possível gravar uma faixa de áudio, igualzinho o gravador do Windows. A melhor parte é que isso é possível também no *mobile*. Para cada plataforma o formato de gravação é diferente – como no Windows, por exemplo, em que o formato de gravação é salvo em WAV, enquanto no iOS o formato é CAF e no Android, 3GP.

Para realizar uma gravação em seu dispositivo, inicialmente siga esses passos:

1. Declare na *Uses* as *units FMX.Media* e *System.IOUtils*.
2. Declare como pública uma variável do tipo *TAudioCaptureDevice* – neste exemplo chamo-a de *Mic*.
3. Crie um botão para iniciar a gravação e outro para interromper.

Com o seu layout pronto, programe o botão para iniciar a gravação da seguinte forma:

```
procedure TForm1.RecordButtonClick(Sender: TObject);
begin
  Mic := TCaptureDeviceManager.Current.DefaultAudioCaptureDevice;
  if Mic <> nil then
  begin
    Mic.FileName := IncludeTrailingPathDelimiter(TPath.GetDocumentsPath) + 'voz.3gp';
    Mic.StartCapture;
  end
  else
  begin
```

```
ShowMessage('Captura de áudio não disponível');
end;
end;
```

Como você pode perceber, o objeto *Mic* é instanciado para o dispositivo padrão de captura de áudio, já encapsulado no FireMonkey. O nome do arquivo é definido, por isso o *Uses* na *Unit System.IOUtils*, pois ela é que dá acesso ao disco. Por fim, o método *StartCapture* inicia enfim a captura de áudio.

O código do botão de *Stop* ficará da seguinte forma:

```
procedure TForm1.StopButtonClick(Sender: TObject);
begin
    if (Mic <> nil) and (Mic.State = TCaptureDeviceState.Capturing) then
    begin
        Mic.StopCapture;
        StopButton.Enabled := false;
        RecordButton.Enabled := true;
    end;
end;
```

Simplesmente verifica se o objeto está instanciado e se o dispositivo de captura está gravando. Em caso positivo, interrompe a gravação através do método *StopCapture*. No final ativa o botão de gravação novamente.

Se você ficou interessado em captura de áudio e vídeo, este link pode ser uma boa alternativa:

[http://docwiki.embarcadero.com/RADStudio/XE7/en/Video\\_Capturing](http://docwiki.embarcadero.com/RADStudio/XE7/en/Video_Capturing)



## 17. Arquivos INI

---

Você pode pensar que o SQLite é o máximo, e é mesmo, mas será que vale a pena guardar tudo em um banco de dados? Às vezes uma configuração que antecede o banco necessita que algo fique em um arquivo separado, às vezes até mesmo para saber em qual banco conectar. Bom, pelo menos esta é a vida comum de um programador Delphi<sup>®</sup> para Windows. Se você sentiu falta do arquivo INI, então este é o seu capítulo.

### ***TMemIniFile***

Para que você usufrua da estrutura fácil e prática de manipulação de arquivos INI, o primeiro passo é declarar a *unit* que implementa essa maravilha, ou seja, *System.IniFiles*.

```
Uses  
System.IniFiles;
```

Porém, no FireMonkey há uma pequena mudança. Para que seu código funcione, independentemente da plataforma, você deve criar um objeto da classe *TMemIniFile*.

Exemplo: que tal guardar um *HighScore*, talvez em um jogo seu feito em Delphi<sup>®</sup> a partir do FireMonkey? O *TMemIniFile* é ideal para isso, pois, além de guardar tudo na memória, é possível que você escreva o arquivo fisicamente, na *sandbox* do seu aplicativo.

```
procedure TForm1.GameOver;  
var  
  IniFile: TMemIniFile;  
Begin
```

```

IniFile := TMemIniFile.Create(IniFileName);
IniFile.WriteInteger('Jogo','Best',BestScore);
IniFile.Free;

End;

```

O seu *IniFileName* virá do disco e para isso é necessário que você declare a *unit System.IOUtils*. Assim, você poderá buscar o caminho para a variável da seguinte forma:

```

IniFileName := System.IOUtils.TPath.GetDocumentsPath +
System.SysUtils.PathDelim + 'pontos.dat';

```

Para ler de um arquivo INI, o procedimento é igual a um aplicativo Windows:

```

IniFile := TMemIniFile.Create(IniFileName);
BestScore := IniFile.ReadInteger('Jogo','Best',0);
IniFile.Free;

```

Para saber um pouco mais sobre o gerenciamento do disco e verificar quais funções são suportadas no Delphi<sup>®</sup> através do *System.IOUtils*, veja o seguinte endereço: [http://docwiki.embarcadero.com/RADStudio/XE7/en/Disk\\_And\\_Directory\\_Support\\_Routines](http://docwiki.embarcadero.com/RADStudio/XE7/en/Disk_And_Directory_Support_Routines)

# 18. Monetizar sua Aplicação

---

Você provavelmente teve notícias de alguns sujeitos, para os quais ninguém dava nada no ensino médio, que se tornaram milionários da noite para o dia com anúncios em seus aplicativos, não é mesmo!? Eu pelo menos conheço um, e você?

O intuito deste capítulo é demonstrar que o Delphi<sup>®</sup> também tem suporte ao AdMob (Android) e ao iAd (iOS). Quem sabe, com um bom aplicativo, você não ganhe um bom dinheiro?

## AdMob

O AdMob é o serviço de anúncios do Google. Com ele seus aplicativos podem exibir anúncios e você pode ganhar algum dinheiro. Para isso é necessário que você crie seu registro e garanta o seu *unique id* para implementar no Delphi<sup>®</sup>. Você pode acessar o site do AdMob e registrar sua conta aqui: <http://www.google.com/admob/>.

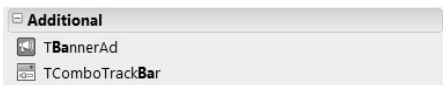
## iAd

O iAd é o serviço de anúncios da Apple. Com ele seus aplicativos podem exibir anúncios. Em um sistema semelhante ao do Google, também requer registro. Você pode conferir mais informações aqui: <https://developer.apple.com/support/appstore/iad-app-network/>.

## TBannerAD

Disponível desde a versão XE6, este é o componente que encapsula o acesso ao AdMob e ao iAd, tornando praticamente transparente para que você programe o mínimo possível. Para aprendizado, utilizaremos o Android como plataforma de destino, pois oferece anúncios de teste.

Inicie um novo projeto, *blank application*, e insira um componente *TBannerAD* em seu formulário.



**Figura 82 – Componente TBannerAd**

Geralmente os anúncios ficam bem abaixo da tela. Para manter esse padrão, você pode alterar a propriedade *Align* do seu *bannerAd* para *MostBottom*.

Antes de prosseguir, caso opte por não realizar esse exemplo com anúncios de teste, recomendo que você crie seu bloco de anúncios no AdMob. É necessário criar uma conta no Google AdSense ou Google Adwords, ou em ambos. Após criar sua conta, crie um bloco de *ads*.

Com seu *ad unit id* em mãos, configure no *OnCreate* do seu formulário o componente com o seguinte código:

```
BannerAd1.AdUnitID := 'SeuAdUnitID';
```

**Nota: para que tudo funcione, você deve ir antes em *Project > Options > Entitlement List* e selecionar “AdMob Service”, marcando a opção como *True*.**

Para colocar um anúncio de seu serviço de publicidade, chame a função *LoadAd*. O procedimento é assíncrono e retorna para os seguintes eventos:

- ***OnDidFail*** – Quando o anúncio não é exibido por algum motivo, retorna uma mensagem de erro.
- ***OnDidLoad*** – Quando tudo funciona conforme o planejado e o anúncio é exibido.

Programa o evento do seu *TBannerAd*, o *OnDidFail*, para que seja exibida uma mensagem caso ocorra um erro, porém antes crie uma variável com nome de *erro*, do tipo *string*, na seção *private* do seu formulário e depois codifique o seguinte:

```

Procedure TForm1.BannerAd1DidFail(Sender : TObject; const Error: String);
Begin
    sErro:= Error;
End;

```

Veja que algumas opções de acesso são necessárias, porém essas marcações são realizadas no *Uses Permissions*. Habilite *ACCESS\_NETWORK\_STATE* e *INTERNET*.

Para testar seu aplicativo, utilize a chamada *LoadAd* no *OnShow* do seu formulário:

```

procedure TForm1.FormShow(Sender: TObject);
begin
    if BannerAd1.AdUnitID = '' then
        begin
            BannerAd1.AdUnitID := 'seu id';
            BannerAd1.LoadAd;
        end;
end;

```

Para maiores informações sobre anúncios, verifique os links:

- [http://docwiki.embarcadero.com/RADStudio/XE7/en/Adding\\_Advertising\\_to\\_Your\\_Mobile\\_Apps#Loading\\_an\\_Ad](http://docwiki.embarcadero.com/RADStudio/XE7/en/Adding_Advertising_to_Your_Mobile_Apps#Loading_an_Ad)
- [http://docwiki.embarcadero.com/RADStudio/XE7/en/Using\\_the\\_AdMob\\_Service](http://docwiki.embarcadero.com/RADStudio/XE7/en/Using_the_AdMob_Service)
- [http://docwiki.embarcadero.com/RADStudio/XE7/en/Using\\_the\\_iAd\\_Service](http://docwiki.embarcadero.com/RADStudio/XE7/en/Using_the_iAd_Service)
- [http://blog.marcocantu.com/blog/2014\\_may\\_using\\_admob\\_android.html](http://blog.marcocantu.com/blog/2014_may_using_admob_android.html)

# 19. Publicando seu Aplicativo

---

Volte para este capítulo quantas vezes achar necessário, pois aqui você terá a informação necessária para realizar o *deploy* e a publicação de seus aplicativos nas lojas *Google Play* e *App Store*.

Antes de realizar a publicação, você viu que é necessário criar uma conta de desenvolvedor tanto para a Apple quanto para o Google. Ambas as contas que permitem a publicação custam algum dinheiro, portanto, prepare o bolso.

Registro feito, pagamento realizado, hora de realizar o *deploy* e publicar o seu aplicativo.

## ***Deployment Manager* – Preparando seu aplicativo para publicação**

O *Deployment Manager* permite realizar *deploy* em seus aplicativos para qualquer plataforma de destino suportada.

**Nota: você deve construir (*build*) o seu projeto antes de realizar o *deploy*.**

No Delphi<sup>®</sup>, selecione *Project > Deployment* para abrir o gerenciador de implementação e siga esses passos:

1. Se você está realizando o *deploy* para Windows, Mac OS X ou iOS:
  - Habilite a conexão com a máquina de destino usando o comando *Connect to Remote Machine*.
  - Clique no botão “Reconcile” para atualizar a lista de arquivos.
2. Adicione seus arquivos adicionais que exigem *deploy* e configure-os, se necessário.

### 3. Clique no botão “Deploy” na *toolbar* do *Deployment Manager*.

O processo de *deploy* começa e o Delphi<sup>®</sup> inclui os arquivos selecionados para o seu aplicativo. Se o *deploy* for bem-sucedido, será exibida uma janela com uma mensagem de sucesso.



Figura 83 – *Deployment Manager*.

## Recursos do *Deployment Manager*



Figura 84 – *Toolbar* do *Deployment Manager*.

|                                    |  |
|------------------------------------|--|
| <b>Botão Reconcile</b>             | Recarrega todas as informações do projeto. Este comando é muito semelhante a um comando de atualização.  |
| <b>Botão Add Files</b>             | Permite adicionar arquivos que não fazem parte do projeto para a lista de implantação. Um exemplo comum seria a adição de arquivos de dados ou arquivos de configuração de que o aplicativo faz uso.   |
| <b>Botão Delete Selected Files</b> | Permite excluir arquivos que foram adicionados anteriormente através do botão <i>Add Files</i> . Observe que você não pode apagar arquivos que o IDE adiciona automaticamente. Ao selecionar esses arquivos, o botão <i>Delete</i> será desativado (em cinza). |
| <b>Botão Add Featured Files</b>    | Exibe uma caixa de diálogo que contém uma lista de arquivos que você pode selecionar para adicionar ao seu projeto. Este comando é usado principalmente para aplicações de banco de dados relacionados.  |
| <b>Botão Deploy</b>                | Implanta os arquivos para a máquina remota. Este comando utiliza a configuração selecionada  |

|                                |   |
|--------------------------------|---|
|                                | para implantar os arquivos. Os arquivos na lista também são implantados automaticamente executando o projeto no Delphi.   |
| <b>Botão Connect</b>           | Este é um botão de dois estados. O botão <i>Connect</i> permanecerá apagado enquanto as máquinas estão conectadas. Quando a conexão falhar ou for bem-sucedida, o botão permanecerá pressionado. Se a operação de conexão for completada com sucesso, os campos de status são atualizados. Caso contrário, uma mensagem de erro é exibida. O botão <i>Connect</i> é desativado quando todas as configurações forem selecionadas, pois você só pode conectar-se a uma máquina de cada vez. |
| <b>Botão Revert to Default</b> | Reverte todas as mudanças que você já fez no <i>Deployment Manager</i> para o projeto atual. O projeto é revertido para o estado em que se encontrava quando o abriu pela primeira vez no <i>Deployment Manager</i> . A opção <i>Keep added files</i> permite que você mantenha os arquivos que adicionou manualmente para o seu projeto. A opção é selecionada por padrão.   |

Para certas plataformas de destino, o *Deployment Manager* adiciona um arquivo de símbolos de depuração remoto (.rsm) para a lista de arquivos prontos para implantar. O arquivo (.rsm) só é necessário durante a execução do projeto no modo de depuração. Por padrão, um arquivo (.rsm) é gerado quando a plataforma de destino selecionada é de 64-bit do Windows, Mac OS X ou *iOS Simulator*. Para ativar ou desativar a geração de um arquivo (.rsm), você pode alternar a opção de símbolos de depuração remota. Ver em: *Project > Options > Delphi Compiler > Linking*.

**Nota: o arquivo (.rsm) só é necessário se você está planejando executar o aplicativo no modo de depuração. Caso contrário, você pode removê-lo da lista.**



## Publicando seu aplicativo na App Store

Após realizar seus testes através do *iOS Simulator* e dos testes em seus dispositivos físicos, chega a hora de enviar para a loja aquele seu aplicativo tão especial.

Para isso é necessário enviar através do *iTunes Connect*. Para iniciar o envio de um aplicativo à loja, abra o site do *iTunes Connect* e faça login com sua conta de desenvolvedor. Acesse a opção “Manage Your Apps”.

Antes de enviar o pacote para a Apple, é necessário preencher um formulário com todos os dados do aplicativo que será publicado. Clique em “Add New App”. Na tela a seguir serão exigidos alguns dados. Em “Default Language”, selecione o idioma principal em que seu aplicativo foi desenvolvido. Em “App Name” defina o nome do aplicativo que será enviado. O terceiro campo pede que seja criado um código único SKU para identificar o aplicativo. Você pode criar qualquer código utilizando letras e números. Por fim, selecione uma *Bundle ID* para seu aplicativo. Se você ainda não possui uma, pode criar uma nova ID no *iOS Dev Center*. Quando todas as informações forem preenchidas corretamente, clique em “Continue”.

Preencha todos os dados corretamente, de acordo com o que for solicitado e não falsifique as informações, do contrário a Apple irá recusar seu aplicativo.

Certifique-se de que você possui uma chave de acesso (*KeyChain*) – sem ela não será possível o envio.

No Delphi<sup>®</sup>, selecione o seu projeto e, no *Project Manager*, expanda a pasta *Configuration*. Marque com um duplo clique a opção “Application Store”.

Você deve configurar várias opções de projetos para seu App iOS antes de criar o aplicativo para distribuição. As propriedades que você configura são empacotadas com o seu aplicativo, geralmente sob a forma de arquivos (.plist). Você não pode modificar essas propriedades depois que construir e assinar seu aplicativo. Essas propriedades fornecem informações essenciais sobre a sua aplicação. Você deve garantir que elas contêm os valores corretos antes de implantar seu aplicativo, ou pode ter que reconstruir a sua aplicação, a fim de alterar os valores configurados.

Para configurar seu aplicativo iOS, selecione *Project > Options*:

1. Na página *Application page*, fornecer os ícones e imagens para representar a sua aplicação.
2. Na página *Entitlement List*, atribuir os direitos (direitos de acesso ao sistema) que você deseja para a sua aplicação.
3. Na página de informações da versão, definir as chaves *Info.plist* do seu aplicativo iOS. Por exemplo: *CFBundleIdentifier* é um identificador exclusivo para a sua aplicação.
4. Você já deve ter um *ID App* para sua aplicação. Você deve fornecer aqui a segunda parte do seu ID de aplicativo, a *ID bundle*. Por exemplo: *com.my-company.myapp*.
5. *CFBundleVersion* é a versão do seu aplicativo. Por exemplo: 1.0.0.
6. *UIDeviceFamily* determina o tipo de dispositivo iOS que seu aplicativo suporta: iPhone, iPad ou iPhone e iPad.

Na IDE, vá em *Project > Build* para compilar seu projeto. Em seguida, clique em “Deploy” para juntar todos os recursos.



**Figura 85 – Opção *Application Store*.**

**Nota:** a primeira vez que você usar uma conta de usuário em seu Mac para executar um aplicativo em um dispositivo iOS, seu Mac irá pedir-lhe para permitir o Delphi® a assinar aplicativos usando sua chave privada. Você deve aceitar isso no seu Mac para que Delphi® possa realizar o *deploy* na sua aplicação.

Sua aplicação, gerada no Mac, é então copiada para o seu sistema de desenvolvimento. Você pode encontrar o seu arquivo de aplicação final (.ipa) na pasta do seu projeto: < pasta do seu projeto > \ iOSDevice \ < Configuração > \ < Nome do Projeto > .ipa.

Por exemplo: C: \ Users \ < usuário > \ Documents \ Embarcadero \ Studio \ Projects \ ProjectFolder \ iOSDevice \ Debug \ ProjectName.ipa.

1. No Mac, execute o aplicativo *Application Loader*.
2. No *Application Loader*, selecione “Entregar App”.
3. Na caixa de combinação, selecione a entrada de aplicativo criada anteriormente no *iTunes Connect*.
4. Clique no botão “Escolher” e navegue até o arquivo do aplicativo (.ipa) que você criou anteriormente.
5. Clique em “Enviar”.

**Dica: você não precisa trazer o arquivo do seu PC de desenvolvimento para o seu Mac. Você pode encontrar a cópia original do arquivo do aplicativo no Mac, dentro do diretório *scratch*.**

O *Application Loader* começa a primeira rodada de testes automatizados em sua aplicação. Se o seu aplicativo passa nessa primeira rodada de testes, o *Application Loader* prossegue para enviar sua aplicação para a Apple.

Quando a Apple recebe o arquivo, eles executam em sua extremidade mais uma rodada de testes automatizados em sua aplicação. Se o seu aplicativo passa nessa segunda rodada de testes, a Apple coloca sua aplicação em uma fila para o teste manual.

Eventualmente, a Apple analisa a sua aplicação manualmente.

Quando seu aplicativo passa ou falha em qualquer um desses testes, você recebe uma notificação por e-mail.

Se o seu aplicativo falhar em algum teste, a apresentação de seu aplicativo para a App Store é abortada. Você deve corrigir o erro relatado e repetir os passos. Por exemplo, se a Apple diz que suas entradas de pacote não correspondem, verifique se você configurou em *Project > Options > Version Info* um valor para a chave

*CFBundleIdentifier*, tendo que ser igual ao ID que você criou para sua aplicação na App Store.

Se o seu aplicativo passa em todos os testes, ele é publicado na App Store, ou programado para ser lançado no futuro (se você configurou para isso).

## Publicando seu aplicativo na Google Play

Seu aplicativo Android é instalado quando você o executar em um dispositivo Android. Você também pode usar o *Deployment Manager* para gerenciar os arquivos instalados. Depois que seu aplicativo Android estiver pronto para implantação final, você pode continuar a construir e **assinar digitalmente** seu aplicativo (a assinatura será explicada no decorrer deste capítulo).

Você deve configurar várias opções para o seu aplicativo Android antes de criar o aplicativo para distribuição. As propriedades que você configura são empacotadas com a sua aplicação no arquivo *AndroidManifest.xml*. Você não pode modificar essas propriedades depois que construir e assinar seu aplicativo, pois elas fornecem informações essenciais sobre a sua aplicação. Você deve garantir que elas contêm os valores corretos antes de realizar o *deploy* em seu aplicativo, ou você pode ter que reconstruí-lo, a fim de alterar os valores configurados.

Quando você cria um aplicativo para a plataforma de destino Android pela primeira vez, o Delphi<sup>®</sup> adiciona um arquivo para a pasta do projeto: *AndroidManifest.template.xml*. Sempre que você criar seu aplicativo para Android, o Delphi<sup>®</sup> lê esse arquivo, substitui alguns espaços reservados no arquivo com valores reais de suas opções de projeto e escreve o conteúdo resultante em um arquivo de saída: *Android \ <build configuration> \ AndroidManifest.xml*. Esse arquivo de saída é o *AndroidManifest.xml* que está incluído no pacote do Android que o Delphi<sup>®</sup> gera quando você instala seu aplicativo para a plataforma Android de destino.

Para personalizar o arquivo *AndroidManifest.xml* de um único projeto, basta editar o conteúdo de *AndroidManifest.template.xml*.

Depois de configurar o seu ambiente para desenvolvimento de aplicativos Android, você pode instalar seu aplicativo como um **pacote assinado**. Posteriormente, você pode fazer o *upload* do **pacote assinado** para lojas de aplicativos como Google Play ou Amazon.

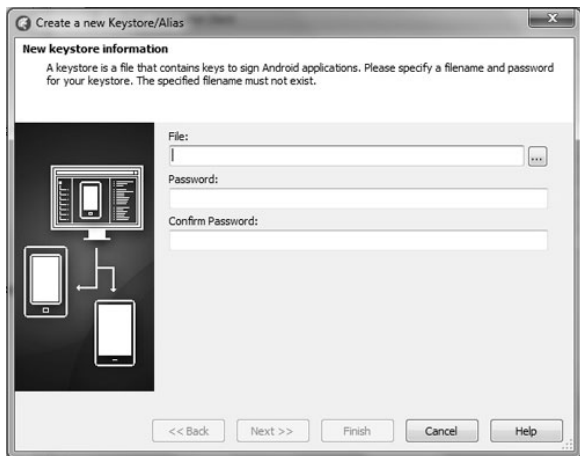
Antes de realizar o último *build*, é necessário criar uma chave privada, chamada de *KeyStore*. Embora não seja recomendável, é possível enviar um aplicativo não assinado para o Google Play, porém é bom que você assine o seu aplicativo para garantir a sua autenticidade como desenvolvedor.

Para criar uma nova *KeyStore*, vá até o menu de opções em: *Tools > Options > Provisioning* e selecione “Android – Application Store”.

Agora basta criar um novo arquivo de armazenamento de chaves onde você registra os certificados de assinatura de código para seus aplicativos.

Nessa página, você fornece informações sobre o seu novo arquivo de armazenamento de chaves.

- **File** – O caminho onde você quer salvar o seu novo arquivo de armazenamento de chaves.
- **Password** – A senha para proteger o arquivo de armazenamento de chaves.
- **Confirm Password** – Repita a senha para o arquivo de armazenamento de chaves.
- **Alias** – Um apelido que identifica exclusivamente o seu novo certificado.
- **Organizational Unit** – O nome do seu departamento dentro da sua organização.
- **Organization** – O nome da sua organização (empresa).
- **City or Locality** – O nome do lugar onde você mora.
- **State or Province** – O nome do estado ou província onde a sua cidade está localizada.
- **Country Code (XX)** – Um código ISO-3166-1 de duas letras que identifica o país onde você mora. Por exemplo: BR.




**Figura 86 – Selecione uma pasta no seu computador para guardar o arquivo.**

Create a new Keystore/Alias

**New key alias information**

A key alias wraps a key pair (a public key and a private key), which is stored as a single element. All keystore entries are accessed via unique aliases. Aliases are case-insensitive.



Alias: WILLIAM

Password: •••••

Confirm: •••••

Validity (days): 9132

First and Last Name: WILLIAM DUARTE

Organizational Unit: IT

Organization: WILLIAM CORP

City or Locality: RIO DE JANEIRO

State or Province: RJ

Country Code (XX): BR

<< Back Next >> Finish Cancel Help

**Figura 87 – Preencha todos os campos com suas informações.**

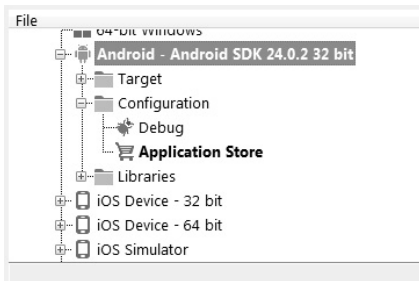


**Figura 88 – Seu certificado para provisionamento está feito.**

Para gerar um pacote assinado contendo seu aplicativo, faça o seguinte:

1. No *Project Manager*, expanda o nó *Build Configurations* e clique duas vezes na configuração *Application Store*.





**Figura 89 – Application Store.**

2. Selecione *Project > Options > Provisioning* e verifique se todos os campos estão preenchidos com os valores corretos. Um arquivo de armazenamento de chaves e um certificado são obrigados a assinar seu aplicativo.
3. Selecione *Project > Deployment* e clique no botão “Deploy” para realizar o *deploy* do seu aplicativo.

Você pode encontrar o pacote de inscrição assinado (.apk) na pasta do projeto:

<Pasta do projeto> \ Android \ <configuração selecionada> \ <Nome do Projeto> \ bin \ <Nome do Projeto>.apk

Vá até o *Developer Console* através do website do *Google Play*, envie o arquivo (.apk) para a loja de aplicativos e sucesso!

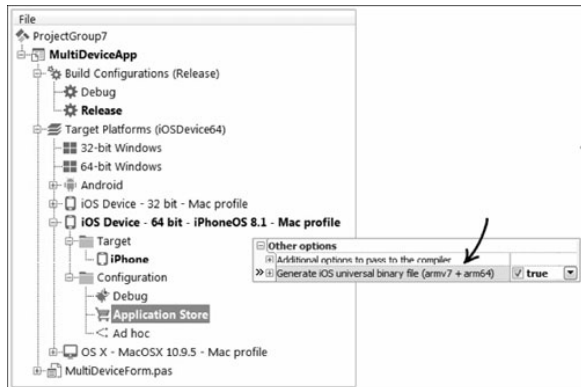
# Apêndice. Delphi XE8 – Visão Geral

---

Com lançamento em 2015, o Delphi<sup>®</sup> XE8 veio com diversas novidades e melhorias, tornando-o uma das melhores versões de todos os tempos. Devido à retrocompatibilidade das versões Delphi<sup>®</sup> e à preocupação da Embarcadero em manter o legado em funcionamento, a maioria dos exemplos anteriores funciona perfeitamente no Delphi<sup>®</sup> XE8. Todos os capítulos foram adaptados para o funcionamento em ambas as versões XE7 e XE8. Neste apêndice, apenas destaquei algumas novidades e um pouco sobre o que há de novo na ferramenta e por que você deve migrar!

## O que há de novo?

A versão XE8 do Delphi<sup>®</sup> agora integra um poderoso compilador *64-bits* para iOS, uma vez que, para que um novo aplicativo seja publicado na Apple Store, ele deve ser compilado na arquitetura de *64-bits*.

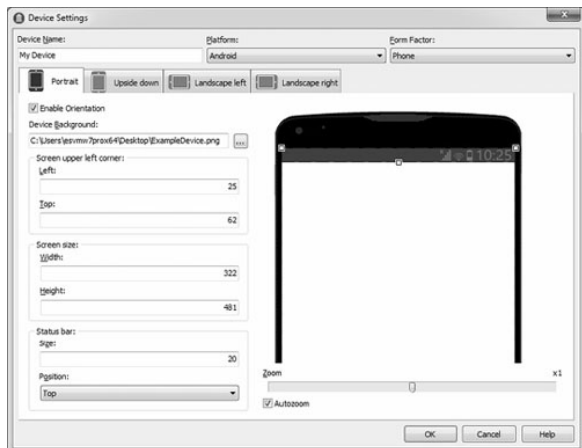


**Figura 90 – Compilador 64-bits.**

O *Multi-device preview* faz com que, em modo de desenho, você obtenha o visual proposto em suas aplicações, como se estivessem instaladas. O Delphi® XE8 vem com uma série de layouts disponíveis e ainda assim permite que o usuário crie seus próprios.



**Figura 91 – Multi-device preview.**



**Figura 92 – Wizard para um novo layout.**

Há novos controles visuais com renderização nativa, como *TMemo*, *TListView*, *TSwitch*, *TEdit* e *TCalendar* – estes, agora nativos para iOS (já eram nativos no Android).

Até a versão XE7, o uso do Google Maps era restrito ao *WebBrowser*, ou melhor, ao componente *TWebBrowser*. Agora, na XE8, você pode utilizar o componente *TMapView*.



**Figura 93 – TMapView.**

Outras novidades importantes são melhorias no suporte a notificações locais, com sons customizados e repetição.

## Novo cliente HTTP

O novo cliente HTTP traz uma implementação nativa dos protocolos HTTP e HTTPS para todas as plataformas (iOS, Android, Windows e OS X), em VCL e FireMonkey. O grande benefício é que essa nova biblioteca é implementada utilizando o SDK de cada plataforma.

## Beacons

Você provavelmente conhece a função de um GPS e no capítulo sobre geolocalização eu explico um pouco sobre como tudo funciona. Tenha em mente que o *beacon* é como um GPS, só que em ambientes fechados. O termo correto seria *Indoor Proximity System*.

Mas como funcionam os *beacons*?

Os *beacons* utilizam *Bluetooth Low Energy* para detectar a proximidade de outros dispositivos e transmitem um número identificador único que é então recebido pelo sistema operacional do dispositivo com o qual ele está se comunicando.

Quando a conexão fica estabelecida, existem duas formas de interagir com os *beacons*:

- Na ação **ativa**, a comunicação se dá através de uma ação do servidor, uma vez que a proximidade de um dispositivo compatível (no caso, o seu *app*) tenha ocorrido. Exemplo: o usuário entra em uma loja e, ao passar pela porta, através do *bluetooth* em seu *smartphone* (o *bluetooth* precisa estar ligado), um *beacon* detecta que o dispositivo atravessou e dispara uma ordem ao servidor, que, por sua vez, retorna com a mensagem: “Promoção de camisas, compre 2 e leve 1” 😊 (licença poética).
- Na ação **passiva** há simplesmente o ato de armazenar, seja localmente ou em um banco de dados, a ocorrência daquela conexão. Por exemplo: o dispositivo Android, modelo Moto G, série 13442, passou por aqui.



Figura 94 – Suporte a *beacons*.

## Castalia

Você pode não conhecer, mas o componente *Castalia*, outrora um componente de terceiros, foi adquirido pela Embarcadero e inserido integralmente na versão

XE8. Com este *add-in* em sua IDE, a programação flui melhor e é possível manter uma melhor endentação. Ele realiza uma análise em seu código-fonte, orientando-o sobre quais trechos de códigos estão mais legíveis que outros. Não é necessário habilitá-lo, ele vem ativado por padrão.



# Considerações Finais

---

Caro leitor, meu muito obrigado por disponibilizar o seu tempo e se dedicar a ler este livro! Foram muitas e muitas horas de estudo e noites em claro para trazer este material até você, sendo você o maior objetivo traçado, desde o início. Eu sinceramente espero que tenha gostado e aprendido a programar para *mobile* utilizando o Delphi<sup>®</sup>.

Os exemplos para os capítulos onde foram utilizados códigos-fonte estão disponíveis no meu site: [www.williamduarte.com.br](http://www.williamduarte.com.br), onde você poderá efetuar o *download* e estudar. Todos foram compilados e testados no Delphi<sup>®</sup> XE7 e XE8. Além do mais, disponibilizei os exemplos na loja do Google Play para uma experiência ainda maior, confira.

Mais uma vez, caro leitor, meu muito obrigado e viva o Delphi<sup>®</sup>!

# Referências Bibliográficas

---

APPLE. **Apple Style Guide**. Apr. 2013. Disponível em: <<http://help.apple.com/asg/mac/2013/>>. Acesso em: 15 jun. 2015.

EMBARCADERO. **System.Sensors.Components.TLocationSensor**. Aug. 18, 2014. Disponível em: <<http://docwiki.embarcadero.com/Libraries/XE7/en/System.Sensors.Components.TLocationSensor>>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Adding a Mac OS X or iOS SDK**. Apr. 20, 2015. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE8/en/Adding\\_a\\_Mac\\_OS\\_X\\_or\\_iOS\\_SDK](http://docwiki.embarcadero.com/RADStudio/XE8/en/Adding_a_Mac_OS_X_or_iOS_SDK)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Android Mobile Application Development**. May 22, 2015. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE8/en/Android\\_Mobile\\_Application\\_Development](http://docwiki.embarcadero.com/RADStudio/XE8/en/Android_Mobile_Application_Development)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **BCCIOSARM.EXE, the C++ Compiler for the 32-bit iOS Device**. Jun. 3, 2015. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE8/en/BCCIOSARM.EXE\\_the\\_C%2B%2B\\_Compiler\\_for\\_the\\_32-bit\\_iOS\\_Device](http://docwiki.embarcadero.com/RADStudio/XE8/en/BCCIOSARM.EXE_the_C%2B%2B_Compiler_for_the_32-bit_iOS_Device)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Connect to SQLite database (FireDAC)**. Aug. 08, 2014. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/Connect\\_to\\_SQLite\\_database\\_\(FireDAC\)](http://docwiki.embarcadero.com/RADStudio/XE7/en/Connect_to_SQLite_database_(FireDAC))>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Creating a Metropolis UI ComboBox**. Jul. 15, 2014. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/Creating\\_a\\_Metropolis\\_UI\\_ComboBox](http://docwiki.embarcadero.com/RADStudio/XE7/en/Creating_a_Metropolis_UI_ComboBox)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Customizing FireMonkey ListView Appearance**. <[http://docwiki.embarcadero.com/RADStudio/XE7/en/Customizing\\_FireMonkey\\_ListView\\_Appearance](http://docwiki.embarcadero.com/RADStudio/XE7/en/Customizing_FireMonkey_ListView_Appearance)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Date and Time Support**. Jul. 4, 2014. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/Date\\_and\\_Time\\_Support](http://docwiki.embarcadero.com/RADStudio/XE7/en/Date_and_Time_Support)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Deploying Your iOS Applications for Submission to the App Store**. Nov. 14, 2014. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/Deploying\\_Your\\_iOS\\_Application\\_for\\_Submission\\_to\\_the\\_App\\_Store](http://docwiki.embarcadero.com/RADStudio/XE7/en/Deploying_Your_iOS_Application_for_Submission_to_the_App_Store)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Deployment Manager**. Nov. 13, 2014. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/Deployment\\_Manager](http://docwiki.embarcadero.com/RADStudio/XE7/en/Deployment_Manager)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **FireMonkey Actions**. Oct. 25, 2013. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/FireMonkey\\_Actions](http://docwiki.embarcadero.com/RADStudio/XE7/en/FireMonkey_Actions)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **FMX.Edit.TCustomEdit.KeyboardType**. Aug. 19, 2014. Disponível em: <<http://docwiki.embarcadero.com/Libraries/XE7/en/FMX.Edit.TCustomEdit.KeyboardType>>. Acesso em: 15 jun. 2015.

EMBARCADERO. **FMX.ListBox.TListBox**. Aug. 14, 2014. Disponível em: <<http://docwiki.embarcadero.com/Libraries/XE7/en/FMX.ListBox.TListBox>>. Acesso em: 15 jun. 2015.

EMBARCADERO. **FMX.Mobile.Controls Sample (Delphi)**. Jul. 11, 2014. Disponível em: <[http://docwiki.embarcadero.com/CodeExamples/XE7/en/FMX.Mobile.Controls\\_Sample\\_\(Delphi\)](http://docwiki.embarcadero.com/CodeExamples/XE7/en/FMX.Mobile.Controls_Sample_(Delphi))>. Acesso em: 15 jun. 2015.

EMBARCADERO. **FMX.MultiView.TMultiView**. Sep. 8, 2014. Disponível em: <<http://docwiki.embarcadero.com/Libraries/XE7/en/FMX.MultiView.TMultiView>>. Acesso em: 215 jun. 2015.

EMBARCADERO.

**FMX.PhoneDialer.IFMXPhoneDialerService.OnCallStateChanged**. Mar. 18, 2013. Disponível em: <<http://docwiki.embarcadero.com/Libraries/XE7/en/FMX.PhoneDialer.IFMXPhoneDialerService.OnCallStateChanged>>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Installing Xcode on a Mac.** Aug. 18, 2014. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/Installing\\_Xcode\\_on\\_a\\_Mac](http://docwiki.embarcadero.com/RADStudio/XE7/en/Installing_Xcode_on_a_Mac)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Joining an iOS Developer Program.** Aug. 18, 2014. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/Joining\\_an\\_iOS\\_Developer\\_Program](http://docwiki.embarcadero.com/RADStudio/XE7/en/Joining_an_iOS_Developer_Program)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Mobile Code Snippets.** Oct. 22, 2014. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/Mobile\\_Code\\_Snippets](http://docwiki.embarcadero.com/RADStudio/XE7/en/Mobile_Code_Snippets)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **PA Server, the Platform Assistant Server Application.** May 29, 2015. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE8/en/PA\\_Server\\_the\\_Platform\\_Assistant\\_Server\\_Application](http://docwiki.embarcadero.com/RADStudio/XE8/en/PA_Server_the_Platform_Assistant_Server_Application)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **SQLite Support in RAD Studio.** Aug. 20, 2014. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/SQLite\\_support\\_in\\_RAD\\_Studio](http://docwiki.embarcadero.com/RADStudio/XE7/en/SQLite_support_in_RAD_Studio)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Tutorial: using FireMonkey layouts.** Jul. 15, 2014. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/Tutorial:Using\\_FireMonkey\\_Layouts](http://docwiki.embarcadero.com/RADStudio/XE7/en/Tutorial:Using_FireMonkey_Layouts)>. Acesso em: 15 jun. 2015.

EMBARCADERO. **Using Custom Format and Parse Expressions in LiveBindings.** May 28, 2014. Disponível em: <[http://docwiki.embarcadero.com/RADStudio/XE7/en/Using\\_Custom\\_Format\\_and\\_Parse\\_Expressions\\_in\\_LiveBindings](http://docwiki.embarcadero.com/RADStudio/XE7/en/Using_Custom_Format_and_Parse_Expressions_in_LiveBindings)>. Acesso em: 15 jun. 2015.

GOOGLE. **Google USB Driver.** Disponível em: <<http://developer.android.com/sdk/win-usb.html>>. Acesso em: 15 jun. 2015.

IOS DEVELOPER LIBRARY. **About Code Signing.** Apple, jul. 23, 2012. Disponível em: <<https://developer.apple.com/library/ios/documentation/Security/Conceptual/CodeSigningGuide/Introduction/Introduction.html>>. Acesso em: 15 jun. 2015.

