

# Tk9.0 em Go: Guia Essencial

por [Wilson M. Faria](#)

## 1. Estrutura Básica (Boilerplate)

Toda aplicação tk9.0 segue esta estrutura fundamental.

```
package main

import (
    // A importação com "." torna as funções do tk9.0 acessíveis diretamente.
    . "modernc.org/tk9.0"
)

func main() {
    // (Opcional) Ative um tema para uma aparência moderna.
    ActivateTheme("clam")
    // --- SEU CÓDIGO DE UI VAI AQUI ---
    l := TLabel(Txt("Olá, Mundo!"))
    Grid(l, Row(0), Column(0), Sticky("we"), Padx("1m"), Pady("1m"))
    // Inicia o loop de eventos. Essencial para a janela aparecer e funcionar.
    App.Wait()
}
```

## 2. Gerenciamento da Janela Principal (App)

Controle a janela principal da sua aplicação.

Função	Exemplo de Uso	Descrição
WmTitle	App.WmTitle("Meu Aplicativo")	Define o título da janela.
SetResizable	App.SetResizable(false, false)	Desabilita redimensionamento (Largura / Altura).
WmGeometry	WmGeometry(App, "800x600+100+50")	Define o tamanho (LxA) e a posição (+X+Y)
Center	App.Center()	Centraliza a janela na tela antes de exibi-la
Wait	App.Wait()	Executa o loop principal da aplicação


Para criar uma janela de tamanho fixo (não redimensionável):

Bloqueie o redimensionamento da janela

```
App.SetResizable(false, false)
WmGeometry(App, "800x600")
```

Unidades de Medida (para Padx, Width, etc.):

Sufixo	Unidade	Exemplo
(nenhum)	Pixels	Width(100)
c	Centímetros	Padx("1c")
m	Milímetros	Pady("5m")
i	Polegadas	Width("2i")
p	Pontos	Font("Arial", "12p")

 Saber disso torna o dimensionamento e o preenchimento muito mais flexíveis e previsíveis.

Por exemplo, Padx("5m") (5 milímetros) terá uma aparência consistente em telas com diferentes densidades de pixels, enquanto Padx(20) (20 pixels) pode parecer grande em uma tela de baixa resolução e minúsculo em uma tela 4K.

### 3. Layout Grid/Pack

- `Pack()` : Empilha verticalmente os widgets um em cima do outro. (Básico)
- `Grid()` : Organiza seus widgets em uma grade de linhas e colunas. (Recomendado)

Funções e Opções Essenciais do Grid:

Opção	Exemplo de Uso	Descrição
Row, Column	<code>Grid(widget, Row(1), Column(0))</code>	Posição do widget na grade (começa em 0).
Columnspan	<code>Grid(widget, Columnspan(2))</code>	Faz o widget ocupar múltiplas colunas.
Rowspan	<code>Grid(widget, Rowspan(3))</code>	Faz o widget ocupar múltiplas linhas.
Sticky	<code>Grid(widget, Sticky("news"))</code>	Alinha/Dimensiona o widget dentro da célula.
Padx, Pady	<code>Grid(widget, Padx("5m"), Pady("2m"))</code>	Adiciona espaçamento externo ao widget.
Ipadx, Ipady	<code>Grid(widget, Ipadx("2m"))</code>	Adiciona espaçamento interno ao widget.
In	<code>Grid(widget, In(meuFrame))</code>	Coloca o widget dentro de um Frame específico.

Valores para Ancoragem de widgets do parâmetro `sticky` (Bússola):

- `"n"` (norte), `"s"` (sul), `"e"` (leste), `"w"` (oeste).
- Combinações: `"nw"` (canto superior esquerdo), `"se"` (canto inferior direito).
- `"ew"` : Estica horizontalmente. `"ns"` : Estica verticalmente.
- `"news"` : Estica em todas as direções para preencher a célula.

⚡ Por padrão, se a célula do grid for maior que o widget dentro dela, o widget ficará centralizado. A opção `Sticky` controla esse alinhamento, "colando" o widget nas bordas da célula. Ela usa as direções da bússola: `"n"` (norte), `"s"` (sul), `"e"` (leste), `"w"` (oeste).

#### Layout Responsivo (essencial quando redimensionável):

Use `GridColumnConfigure` e `GridRowConfigure` para dizer quais linhas/colunas devem se expandir quando a janela for redimensionada.

```
// Faz a coluna 1 e a linha 0 do 'meuFrame' se expandirem.
GridColumnConfigure(meuFrame, 1, Weight(1))
GridRowConfigure(meuFrame, 0, Weight(1))
```

⚡ Esta é a chave para criar layouts adaptáveis. Por padrão, se a janela aumenta, o espaço extra fica vazio porque as linhas e colunas do grid têm um "peso" (`Weight`) zero. Ao atribuir um peso maior que zero a uma linha ou coluna, você está dizendo ao Grid: "Se houver espaço extra disponível, distribua-o para esta linha/coluna proporcionalmente ao seu peso".

### 4. Widgets e Opções Comuns

Widget	Construtor	Opções Comuns
Label	<code>TLabel(...)</code>	<code>Txt</code> , <code>Textvariable</code> , <code>Image</code> , <code>Font</code> , <code>Foreground</code>
Button	<code>TButton(...)</code>	<code>Txt</code> , <code>Command</code> , <code>Image</code>
Entry	<code>TEntry(...)</code>	<code>Textvariable</code> , <code>Width</code>
Checkbutton	<code>TCheckbutton(...)</code>	<code>Txt</code> , <code>Textvariable</code>
Frame	<code>TFrame(...)</code>	<code>Relief</code> , <code>Borderwidth</code> , <code>Padding</code>
Separator	<code>TSeparator(...)</code>	<code>Orient("horizontal" or "vertical")</code>
Progressbar	<code>TProgressbar(...)</code>	<code>Orient</code> , <code>Mode("determinate" or "indeterminate")</code>
Listbox	<code>Listbox(...)</code>	<code>Height</code> , <code>SelectMode("multiple")</code>
Text	<code>Text(...)</code>	<code>Width</code> , <code>Height</code> , <code>Wrap("word")</code>

⚡ `Button(..)` vs `TButton(..)`  
O "T" indica que é um widget "temático" do ttk, que se adapta melhor à aparência do sistema operacional.

## 5. Estilização e Temas

Ação	Exemplo de Código
Ativar Tema	<code>ActivateTheme("clam")</code> , <code>ActivateTheme("alt")</code>
Cor de Fundo	<code>Background("lightblue")</code> OU <code>Background("#ADD8E6")</code>
Cor do Texto	<code>Foreground("red")</code> OU <code>Foreground("#FF0000")</code>
Fonte	<code>Font("Helvetica", 14, "bold")</code>
Relevo da Borda	<code>Relief("sunken")</code> (valores: <code>flat</code> , <code>raised</code> , <code>sunken</code> , <code>ridge</code> , <code>groove</code> )
Largura da Borda	<code>Borderwidth(2)</code>
Configurar um Estilo	<code>StyleConfigure("styleName", Opts)</code>
Aplicar um Estilo	<code>Style("styleName")</code>

### Criando e aplicando estilos

```
ActivateTheme("clam")
StyleConfigure("blue.TButton",
    Background("#b2d2dd"),
    Foreground("#003446"),
    Font("Arial Rounded", "10", "bold"),
)
StyleConfigure("Blue.Horizontal.TProgressbar",
    Background("#003446"),
    Troughcolor(White),
    Bordercolor("#42798b"),
    Lightcolor("#add8e6"),
    Darkcolor("#003446"),
)
b := TButton(Txt("Selecionar Diretório"), Style("blue.TButton")) // Usando um Style
pb := TProgressbar(Mode("determinate"), Maximum(5), Style("Blue.Horizontal.TProgressbar")) // Usando um Style
121 := TLabel(Textvariable("Aguardando diretório"), Background("#add8e6")) // Diretamente na instância
```

⚡ Alguns Widgets recebem definições de `Background` / `Font` / `Foreground` quando são instanciados, mas outros precisam ser configurados usando `Style` e um estilo customizado com `StyleConfigure`

## 6. Dados Dinâmicos e Interação

### 1) Capturar dados com `Textvariable()`

É um mapeamento interno de string usado para exibir e atualizar dados dinamicamente em widgets (Label, Entry, etc.).

```
inputUsuario := TEntry(Textvariable("Digite o usuário.."))
```

Isso cria um campo de texto já inicializado com "Digite o usuário..".

Mais tarde você pode acessar o valor atual com:

```
usuario := inputUsuario.Textvariable()
```

### 1) Atualizar dados com `Configure()`

O método `Configure` permite mudar opções de um widget já existente.

Quando usado com `Textvariable`, ele troca o valor exibido:

```
status := TLabel(Textvariable("Aguardando..."))
// mais tarde, dentro de um Command ou goroutine
status.Configure(Textvariable("Processando..."))
```

## 7. Diálogos Nativos do Sistema

Use para uma experiência de usuário consistente com o SO.

Diálogo	Função	Exemplo de Uso
Abrir Arquivo	<code>GetOpenFile(...)</code>	<code>caminho := GetOpenFile(Title("Abrir..."), Filetypes(tipos))</code>
Selecionar Pasta	<code>ChooseDirectory(...)</code>	<code>pasta := ChooseDirectory(Title("Escolha uma Pasta"))</code>
Caixa de Mensagem	<code>MessageBox(...)</code>	<code>MessageBox(Title("Aviso"), Msg("Operação concluída."), Icon("info"))</code>

### Opções Comuns para Diálogos:

- `Title("...")` : Título da MessageBox.
- `Msg("...")` : Mensagem principal central da MessageBox.
- `Icon("info" | "warning" | "error" | "question")` : Ícone da MessageBox.
- `Type("ok" | "okcancel" | "yesno")` : Botões da MessageBox.
- `Detail("...")` : Espaço extra para texto informativo na MessageBox

```
seguir := MessageBox(Type("yesno"), Icon("question"),
    Title("Confirmar Inicio do Processo"),
    Msg("Processeguir com o processamento dos dados?"))
```

## 8. Barra de Progresso (Progressbar)

A Progressbar (TProgressbar) é usada para indicar andamento de uma tarefa. No Tk9.0, ela é temática (ttk) e suporta modos diferentes de operação.

### Criando uma Progressbar simples

```
bar := TProgressbar(
    Orient("horizontal"),      // orientação
    Mode("determinate"),      // ou "indeterminate"
    Maximum(100),              // valor máximo
    Value(0),                  // valor inicial
    Style("Blue.Horizontal.TProgressbar")) // estilo (opcional)
```

### Opções principais

- `Orient("horizontal" | "vertical")` → direção da barra
- `Mode("determinate")` → progride de 0 até um valor definido
- `Mode("indeterminate")` → “anima” indefinidamente (quando não se sabe a duração da tarefa)
- `Maximum(x)` → valor máximo (ex.: 100 para porcentagem)
- `Value(x)` → valor atual (pode ser alterado depois via `.Configure(Value(x))`)
- `Style("...")` → permite personalizar cores e aparência

### Atualizando valores manualmente

```
bar.Configure(Value(50)) // move para 50%
bar.Configure(Value(100)) // completa a barra
```

### Barra indeterminada (animação automática)

Útil quando não sabemos a duração da tarefa.

```
bar := TProgressbar(Mode("indeterminate"))
bar.Start(50) // avança a cada 50ms
// ...
bar.Stop()    // para a animação
```

# 10. Código completo no GitHub + referências oficiais

Quer ver tudo isso funcionando **de ponta a ponta** (channels, goroutines, `NewTicker` , `Progressbar` com estilo, `MessageBox` , validações e bloqueio/desbloqueio de botões)?

➡ Acesse o repositório: `github.com/wilsonmfaria/GoTK9.0App`

## O que você encontra lá:

- Estrutura real de app desktop com **janela fixa**, **tema** `clam` e **layout com** `Grid` .
- **Concorrência na prática**: tarefas longas em **goroutine**, comunicação via `chan []string` , e atualização de UI no **loop do** `NewTicker` .
- **Feedback ao usuário**: `Progressbar` (determinística), **status animado** com “pontinhos” usando `atomic.Value` , e **diálogos de confirmação**.
- **Boas práticas de UX**: **desabilitar/reabilitar** botões durante processamento, validação de entradas (diretório/usuário).
- **Estilização** via `StyleConfigure` (cores, fontes e estilos dedicados para botões e barras de progresso).

## Referências oficiais (para consulta e aprofundamento)

- **Documentação da API (pkg.go.dev)** – lista de funções, opções e widgets:  
<https://pkg.go.dev/modernc.org/tk9.0>
- **Exemplos oficiais (repositório do projeto)** – coleções de exemplos práticos prontos:  
[https://gitlab.com/cznic/tk9.0/-/tree/v1.72.0/\\_examples](https://gitlab.com/cznic/tk9.0/-/tree/v1.72.0/_examples)