**Отчет по Главе 5: Привязка данных (Data Binding)**

**1. Введение**

Данная глава посвящена одной из наиболее мощных и фундаментальных концепций в .NET MAUI — привязке данных (Data Binding). Привязка данных позволяет установить связь между свойством источника данных (обычно это объект ViewModel) и свойством целевого объекта (обычно это элемент пользовательского интерфейса). Значимость этой технологии заключается в том, что она позволяет отделить логику представления от бизнес-логики, упрощает синхронизацию данных между UI и кодом, уменьшает количество шаблонного кода для обновления UI и является основой для паттерна проектирования MVVM (Model-View-ViewModel). Освоение привязки данных необходимо для создания динамичных и легко поддерживаемых настольных и мобильных приложений.

**2. Основная часть**

**2.1. Основные концепции и принципы**

* **Источник привязки (Binding Source):** Объект, предоставляющий данные. В .NET MAUI это часто экземпляр класса ViewModel.
* **Цель привязки (Binding Target):** Элемент пользовательского интерфейса, свойство которого будет связано с источником. Целевое свойство должно быть BindableProperty.
* **BindingContext**: Свойство элемента UI (обычно устанавливается на уровне страницы или корневого контейнера), которое указывает на источник привязки. Все привязки внутри этого элемента (и его дочерних элементов, если они не переопределяют BindingContext) будут разрешаться относительно этого контекста.
* **Расширение разметки {Binding}**: Используется в XAML для объявления привязки. Например, Text="{Binding UserName}".
* **Путь (Path):** Свойство внутри {Binding} (часто опускается, если это единственное значение), которое указывает на конкретное свойство в BindingContext. Например, Text="{Binding User.Name}" или просто Text="{Binding Name}".
* **Режимы привязки (Mode):** Определяют направление потока данных:
  + OneWay: Данные передаются от источника к цели. Изменения в источнике обновляют цель, но не наоборот. Это режим по умолчанию для большинства нередактируемых свойств (например, Label.Text).
  + TwoWay: Данные передаются в обе стороны. Изменения в источнике обновляют цель, а изменения в цели (например, ввод текста в Entry) обновляют источник. Режим по умолчанию для редактируемых свойств (например, Entry.Text, Slider.Value).
  + OneWayToSource: Данные передаются от цели к источнику. Изменения в цели обновляют источник, но не наоборот. Используется редко.
  + OneTime: Данные передаются от источника к цели только один раз, когда устанавливается BindingContext или создается привязка. Последующие изменения в источнике не влияют на цель.
* **INotifyPropertyChanged**: Интерфейс, который должен реализовывать класс-источник (ViewModel), если требуется, чтобы UI обновлялся при изменении его свойств (для режимов OneWay и TwoWay). Он определяет событие PropertyChanged, которое необходимо вызывать при изменении значения свойства.
* **Команды (ICommand):** Механизм для привязки действий пользователя (например, нажатие кнопки) к методам в ViewModel. Свойства элементов управления, такие как Button.Command, привязываются к свойству типа ICommand в ViewModel. CommandParameter позволяет передавать данные в команду.
* **Конвертеры значений (IValueConverter):** Позволяют преобразовывать данные между источником и целью привязки, если их типы или форматы не совпадают.
* **Форматирование строк (StringFormat):** Позволяет форматировать отображаемое значение непосредственно в XAML.

**2.2. Примеры кода с пояснениями**

Для демонстрации привязки данных была создана страница DataBindingDemoPage.xaml и класс SimpleViewModel.cs.

**2.2.1. SimpleViewModel.cs**  
Этот класс выступает в роли источника данных (ViewModel) и реализует INotifyPropertyChanged для уведомления UI об изменениях.

using System.ComponentModel;

using System.Runtime.CompilerServices;

using System.Windows.Input;

namespace HelloMauiApp;

public class SimpleViewModel : INotifyPropertyChanged

{

private string \_userName;

public string UserName

{

get => \_userName;

set

{

if (\_userName != value)

{

\_userName = value;

OnPropertyChanged();

OnPropertyChanged(nameof(Greeting));

}

}

}

public string Greeting => $"Hello, {UserName}!";

// ... другие свойства и команды ...

public ICommand UpdateNameCommand { get; }

public ICommand ResetNameCommand { get; }

public SimpleViewModel()

{

UserName = "MAUI User";

// ... инициализация других свойств ...

UpdateNameCommand = new Command<string>((newName) =>

{

UserName = string.IsNullOrWhiteSpace(newName) ? "Default User" : newName;

});

ResetNameCommand = new Command(

execute: () => { /\* ... \*/ },

canExecute: () => !string.IsNullOrEmpty(UserName) && UserName != "MAUI User"

);

}

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

if (propertyName == nameof(UserName) && ResetNameCommand is Command cmd)

{

cmd.ChangeCanExecute();

}

}

}

*Пояснение:* UserName – это свойство, изменения которого отслеживаются. OnPropertyChanged() вызывается в сеттере. Greeting – вычисляемое свойство, которое также обновляется при изменении UserName. UpdateNameCommand и ResetNameCommand – команды для взаимодействия с UI.

**2.2.2. DataBindingDemoPage.xaml**  
На этой странице демонстрируются различные типы привязок.

<ContentPage.BindingContext>

<local:SimpleViewModel/>

</ContentPage.BindingContext>

<!-- OneWay Binding -->

<Label Text="{Binding UserName}" FontSize="Large"/>

<Label Text="{Binding Greeting}" FontSize="Medium" TextColor="Green"/>

<!-- TwoWay Binding -->

<Entry Text="{Binding EntryText, Mode=TwoWay}" Placeholder="Enter text to update ViewModel"/>

<Label Text="{Binding EntryText, StringFormat='Entry Text is: {0}'}"/>

<!-- Slider Binding (TwoWay по умолчанию для Value) -->

<Slider Value="{Binding SliderValue}" Minimum="0" Maximum="1"/>

<Label Text="{Binding SliderValueText}" HorizontalOptions="Center"/>

<ProgressBar Progress="{Binding SliderValue}"/>

<!-- Command Binding -->

<Entry x:Name="NewNameEntry" Placeholder="Enter new name for greeting"/>

<Button Text="Update Name from Entry"

Command="{Binding UpdateNameCommand}"

CommandParameter="{Binding Source={x:Reference NewNameEntry}, Path=Text}"/>

<Button Text="Reset All" Command="{Binding ResetNameCommand}"/>

<!-- OneTime Binding -->

<Label Text="{Binding UserName, Mode=OneTime}" TextColor="Gray"/>

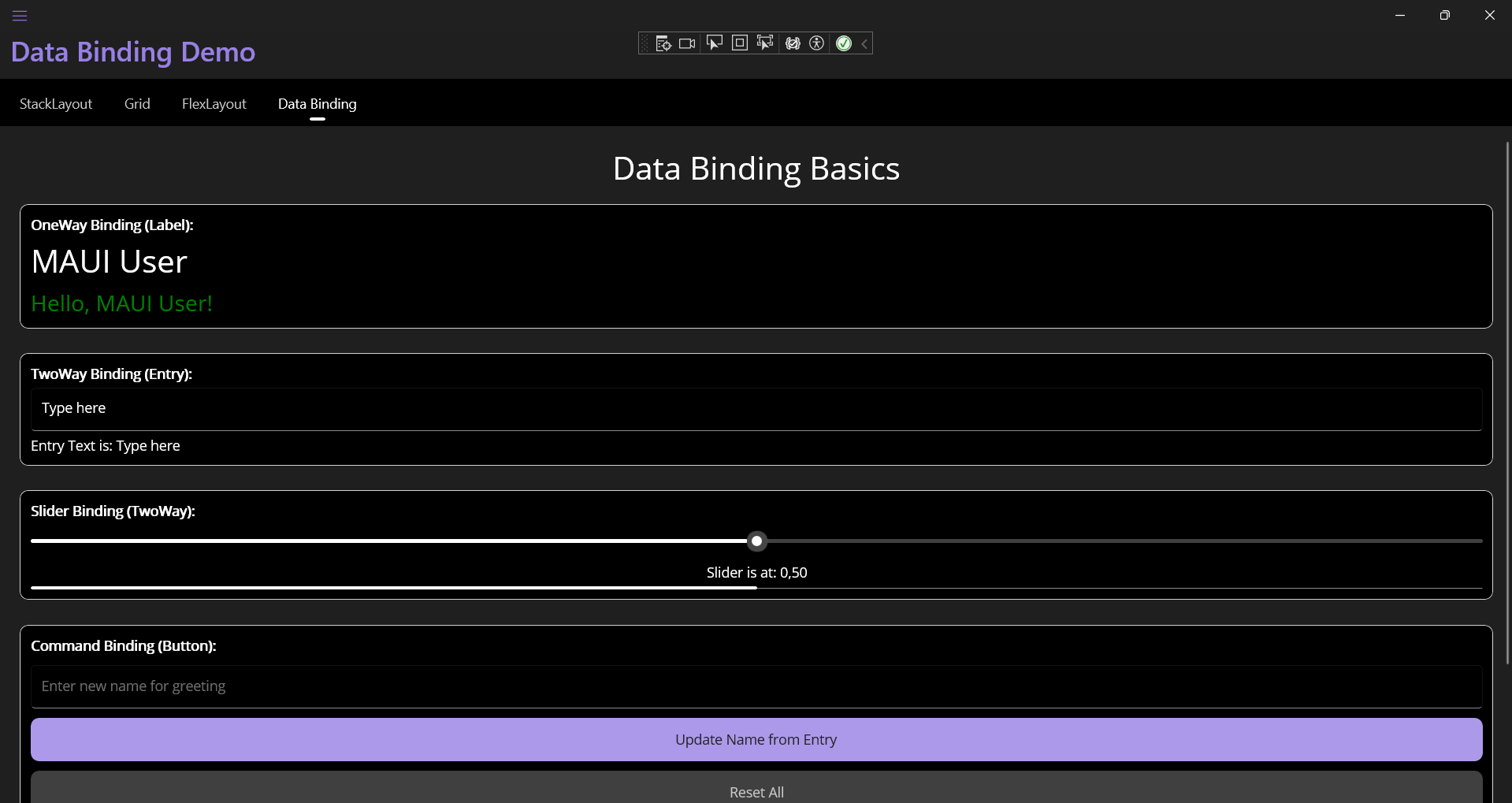
*Пояснение:* BindingContext страницы устанавливается на экземпляр SimpleViewModel. Демонстрируются привязки OneWay для Label, TwoWay для Entry и Slider, привязка ProgressBar к значению Slider, а также привязка кнопок к командам с передачей параметра из другого элемента (NewNameEntry). OneTime привязка показывает, что значение устанавливается только один раз.

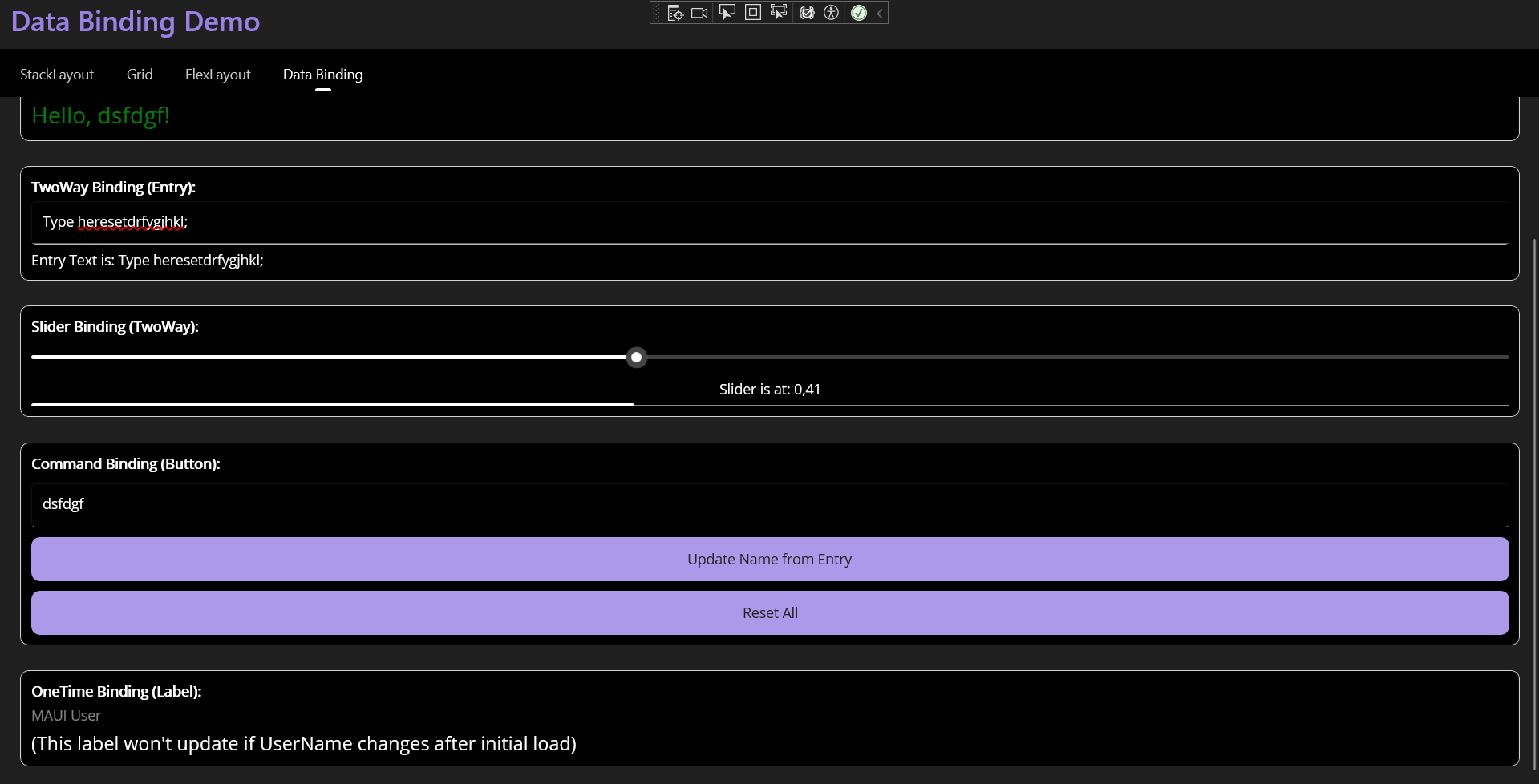
**3. Заключение**

Привязка данных является мощным механизмом в .NET MAUI, который значительно упрощает разработку пользовательских интерфейсов, делая их более отзывчивыми и легко поддерживаемыми. Понимание режимов привязки, реализации INotifyPropertyChanged и использования команд позволяет создавать сложные взаимодействия между UI и бизнес-логикой с минимальным количеством связующего кода. Эта концепция является центральной для паттерна MVVM и имеет прямое применение при разработке настольных приложений, способствуя созданию чистого, тестируемого и масштабируемого кода.

**4. Скриншоты и примеры**

Страница DataBindingDemoPage была создана для демонстрации различных аспектов привязки данных. Приложение было запущено на платформе Windows.



*Рис. 5.1 и 5.2. Демонстрация работы привязки данных на странице DataBindingDemoPage до и после взаимодействия пользователя."*

**Процесс выполнения примеров:**

1. Создан класс SimpleViewModel.cs, реализующий INotifyPropertyChanged и содержащий свойства и команды для привязки.
2. Создана страница DataBindingDemoPage.xaml, на которой BindingContext установлен на экземпляр SimpleViewModel.
3. На XAML-странице размещены различные элементы управления (Label, Entry, Slider, ProgressBar, Button), свойства которых привязаны к свойствам SimpleViewModel с использованием разных режимов привязки и команд.
4. Страница добавлена в навигацию AppShell.xaml.
5. Проведено тестирование:
   * Изменение текста в Entry приводило к обновлению связанного Label (TwoWay).
   * Изменение положения Slider обновляло связанный Label и ProgressBar (TwoWay).
   * Нажатие кнопки "Update Name from Entry" с введенным текстом обновляло Label с приветствием (Command с параметром).
   * Кнопка "Reset All" сбрасывала значения в ViewModel и обновляла UI, а ее доступность управлялась через CanExecute команды.
   * Label с OneTime привязкой не изменялся после первоначальной загрузки.  
     Все привязки работали корректно, демонстрируя синхронизацию данных между UI и ViewModel. Проблем при выполнении примеров не возникло.

**5. Документация**

* **Ссылка на GitHub с исходным кодом:** https://github.com/RaniZee/.NET-MAUI-Showcase-Cross-Platform-Applications
* **Описание выполненных шагов:**
  1. Изучены материалы по теме "Привязка данных" (глава 5) с ресурса [metanit.com/sharp/maui/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fmetanit.com%2Fsharp%2Fmaui%2F).
  2. Создан класс SimpleViewModel с реализацией INotifyPropertyChanged и несколькими свойствами и командами.
  3. Создана страница DataBindingDemoPage с установленным BindingContext и элементами, привязанными к ViewModel.
  4. Продемонстрированы режимы привязки OneWay, TwoWay, OneTime.
  5. Реализована привязка к командам ICommand, включая передачу параметров и управление CanExecute.
  6. Протестирована работоспособность всех привязок на платформе Windows.
  7. Подготовлены скриншоты для отчета.