# List View and Table View

List Views and Table Views allow the display of a collection in a consistent way. Each line item in the view is associated with some kind of template that displays properties of a linked object from the collection. This works in a similar way to how the Android SDK works with these kinds of objects.

# List View

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/listview/>

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/collectionview/>

A list view has an **item source** property which is the collection the list will be displaying. This can be any kind of collection that supports enumeration.

The **Item Template** property allows the use of a defined **View Cell** which will define how the cell in the view should look. If this is not set the linked object’s To String will be used.

The **Row Height** property is the height in pixels of each item in the view. Default is big enough for about two labels or so. Doesn’t automatically size, must be set manually if larger desired size.

The **Header/Footer** properties allow either a string or a layout

The **Item Tapped** event allows reaction to taps on specific items. The second parameter to the event handler contains the **Item** property which is a direct reference to the connected object for that row.

## View Cell

This class defines how a row in a list will appear. It has content just like a content page and therefore can have layouts, labels, buttons, etc. Like a content page, the cell contains one View, which of course can be a layout with additional views.

Inheriting from this class to create your own cell type is the preferred method as often there is a fair bit of layout related code that needs to be defined.

When creating the List View, the View Cell is assigned to the Item Template like so:

ItemTemplate = new DataTemplate(typeof(OurCustomCellClass))

View cells can access the **Parent** property to get a reference to the view (usually a List View) that contains them. The parent can then be cast into the appropriate type and worked with. For instance it might be helpful to access the underlying collection of the List View from the cell.

The **Binding Context** property is a reference to the underlying object stored in this cell.

## Adding and Removing Items

While using the list view it might be necessary to add and remove objects from the underlying collection. The list needs to be updated to stay in sync with the collection.

**Note:** When working with a non-static list, choose a collection that is observable. This means that the list view will be notified of changes to the underlying collection. For example, the ObservableCollection data structure works for this. This is available in the System.Collections.ObjectModel namespace.

Using **LINQ (Language Integrated Query)** to remove items from an observable collection will automatically update the List View and keep it in sync with the underlying collection.

## Context Actions and Menu Item for Cells

A View Cell may have a context sensitive menu that will appear when the cell is held (long tap) on Android and swiped left on iOS. Menu items added to the Context Actions list of a cell work like buttons and have a Clicked event hander.

# Table View

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/tableview>

A table view is a view for displaying scrollable lists of data or choices where there are rows that don't share the same template. Unlike ListView, TableView does not have the concept of an ItemsSource, so items must be added as children manually. This means you don’t bind it to an underlying data source.

## Structure of a Table View

* **Table Root**
  + **Table Section(s)** including usually a Title property. There can be multiple sections.
    - **ViewCell(s)** display the individual item(s) in each section

Intent – The intent property is an enumeration indicating the type of use the table will have. It is a good practice to choose the closest fit as that will affect the look and feel on the native platform. The choices are Form, Data, Menu, and Settings. There are subtle differences between each, depending on platform.

## XAML

<TableView Intent="Settings">

<TableRoot>

<TableSection Title="Test">

<SwitchCell Text="This is an on/off" On=”true”/>

<EntryCell Label="Some Label" Placeholder="Placeholder Text"/>

</TableSection>

</TableRoot>

</TableView>

## C# Code

Content = new TableView {

Root = new TableRoot {

new TableSection...

},

Intent = TableIntent.Settings

};

Each table section should have at least one ViewCell added to it.

View Cells can be a built in cell type (TextCell, ImageCell, SwitchCell, or EntryCell) or a custom class.

## Events

Common events to handle from a Table View are the Entry Cell’s Completed event (when the entry is finished) and the OnChanged event of the Switch Cell (occurs when the view is toggled on/off).