webMethods Mobile Designer Native User Interface Reference

Version 10.3

October 2018

WEBMETHODS

Document ID: MD-NUI-103-20191112

Table of Contents

About this Guide....5
Document Conventions....5
Online Information and Support....6
Data Protection....7

Mobile Designer Native User Interface....9
About the Native User Interface (NativeUI) Library....10
Look-and-Feel When Using the NativeUI Library....10
About Using the NativeUI Library....11
Mobile Application Design....12
Hierarchy of NativeUI Objects for a User Interface....13
Setting and Querying NativeUI Object Attributes....14
Handling Events Generated by User Actions....15
Transitioning Between Windows and Views....16
Defining the Layout of Objects in the User Interface....18
Controlling the Inner Padding of Parent Objects....18
Positioning Elements in a Parent Object....20
Sizing Child Elements....22
Controlling the Vertical Spacing Between Child Elements....24
Controlling the Horizontal Alignment of Elements....25
Using Tables to Control the Layout of Elements....26
Managing Object Focus....31
Background Colors and Images....32
Adding Support for Right-to-Left Languages....33
Using Multiple Panes for Tablet User Interfaces....36
Managing the Layout of Panes....37
Designing Applications to Run on Both Tablets and Smaller Devices....38
Determining the Device Size at Run Time....38
Adding Panes to a Window....39
When to Use Views or Panes....40
JavaScript Bridge....41
Maintaining Good Security....41
Sending a Message to JavaScript from Java....42
Evaluating an Arbitrary Chunk of JavaScript Code....42
Sending a Message to Java from JavaScript....42
Using Tabbed Views....43
Integration in Mobile Designer....43
Using List Views and Elements....44
Swiping Behavior....46
Using Edit Mode in List Views....46
Using Element Identifiers....46

Native User Interface (NativeUI) Objects....49
About the NativeUI Objects....51
nUIAlertDialog....51
nUIButtonElement....52
nUICheckboxButton....53
nUIContainerElement....54
nUIDateEntry....55
nUIDialogWindow....56
nUIDisplayObject....57
nUIDropdownlistEntry....58
nUIElementDisplay....58
nUIEntryElement....59
nUIFloatingEntry....60
nUIImageElement....61
nUIListElement....61
nUIListView....62
nUINavbuttonElement....62
nUINavView....65
nUIObject....66
nUIPopupMenuBuilder....66
nUIProgressanimElement....67
nUIRadioCheckbox....68
nUISearchEntry....69
nUISearchNavButton....70
nUISeparatorElement....70
nUISpacerElement....71
nUISwitchButton....71
nUITableButton....72
nUITablecellElement....73
nUITableElement....73
nUITablerowElement....75
nUITabView....75
nUITextfieldElement....76
nUITimerObject....77
nUIViewDisplay....77
nUIWebView....78
nUIWebviewCallBack....80
nUIWebviewElement....80
nUIWindowDisplay....82

About this Guide

This guide describes the Mobile Designer native user interface that you can use to create user interfaces for mobile applications. It contains information for both application designers who want to design user interfaces for mobile applications and developers who want to code user interfaces for mobile applications.

With respect to processing of personal data according to the EU General Data Protection Regulation (GDPR), appropriate steps are documented in *webMethods Mobile Development Help*, *Managing Personal Data*.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format *folder.subfolder.service* , APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.

Convention	Description
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at "http://documentation.softwareag.com". The site requires credentials for Software AG's Product Support site Empower. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to "empower@softwareag.com" with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at "https://empower.softwareag.com/".

You can find product information on the Software AG Empower Product Support website at "https://empower.softwareag.com".

To submit feature/enhancement requests, get information about product availability, and download products, go to "Products".

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the "Knowledge Center".

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at "https://empower.softwareag.com/public_directory.asp" and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at "http://techcommunity.softwareag.com". You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.

- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 Mobile Designer Native User Interface

- About the Native User Interface (NativeUI) Library 10
- Look-and-Feel When Using the NativeUI Library 10
- About Using the NativeUI Library 11
- Mobile Application Design 12
- Hierarchy of NativeUI Objects for a User Interface 13
- Setting and Querying NativeUI Object Attributes 14
- Handling Events Generated by User Actions 15
- Transitioning Between Windows and Views 16
- Defining the Layout of Objects in the User Interface 18
- Managing Object Focus 31
- Background Colors and Images 32
- Adding Support for Right-to-Left Languages 33
- Using Multiple Panes for Tablet User Interfaces 36
- JavaScript Bridge 41
- Using Tabbed Views 43
- Using List Views and Elements 44
- Using Element Identifiers 46

About the Native User Interface (NativeUI) Library

The webMethods Mobile Designer native user interface (NativeUI) library provides a standard way to create user interfaces that match the expected behavior of a platform. For example, you can use the NativeUI library to define a user interface that works equally well on Android and iOS. The resulting user interface typically matches the behavior and look-and-feel that is expected on each target device. For more information, see "Look-and-Feel When Using the NativeUI Library" on page 10.

Mobile Designer is installed with several sample applications, many of which use the NativeUI library. The NativeUI library is made up of several objects. The descriptions of these objects in "Native User Interface (NativeUI) Objects" on page 49 include code samples that illustrate how to use each of the NativeUI objects.

Some NativeUI objects are relatively simple, such as buttons or text entry fields. Others objects are more complex, such as navigation bars or scrollable containers. Each of the NativeUI objects maps to an object on the target device, allowing the user interface to adapt to all target platforms, including devices with touchscreen user interfaces, physical keyboards, and other input methods.

Look-and-Feel When Using the NativeUI Library

You use the NativeUI library to create the user interface for your mobile application. When you compile your application, the NativeUI Class implementation for each platform is replaced by the native version of these classes that will execute when the application is running on the target device.

This class controls how a NativeUI object behaves and looks on the target device. The class to which Mobile Designer translates a NativeUI class depends on whether the NativeUI library has platform-specific support for the platform.

- When the NativeUI library includes platform-specific support for a platform, Mobile Designer translates the NativeUI classes into platform-specific classes. As a result, a user interface object renders as expected on the target device, using the platform look-and-feel and behavior.

  The NativeUI provides support for Android and iOS.

  For example, if you use the NativeUI object nUICheckboxButton and compile your application for an iOS device, Mobile Designer translates NativeUI object nUICheckboxButton to the iOS UISwitch class. As a result, when the user interface displays on the iOS device, it uses the iOS UISwitch class to render the check box.

- When the NativeUI library does *not* include platform-specific support for a platform, Mobile Designer uses a general, graphical skin. In this case, the user interface renders on the target device using a general graphical skin rather than a platform-specific look-and-feel.

The general graphical skin renders all of the available NativeUI objects, including features such as on-screen pop-up keyboards.

NativeUI and Phoney Skins

Phoney is a phone simulator that is not platform-specific. You can use Phoney to test your mobile applications. For more information, see *Using webMethods Mobile Designer*.

When you use the NativeUI library for your mobile application user interface and run the application in Phoney, the look-and-feel for the user interface depends on whether Mobile Designer provides platform-specific skins for the platform you are simulating in Phoney.

- When Mobile Designer includes platform-specific skins for a simulated platform, Phoney renders the user interface using the Phoney platform-specific skin.

  The platform-specific Phoney skins do not provide an exact representation of how the user interface will look on the platform. However, the platform-specific Phoney skins do allow you to get a better idea of how your application's user interface looks in the target platform. The Phoney skins attempt to match a platform's look-and-feel.

- When Mobile Designer does *not* include a platform-specific skin for a simulated platform, Phoney renders the user interface using the general graphical skin.

While developing your mobile applications, using Phoney saves you time because you can use Phoney to quickly visualize your application's user interface rather than having to deploy your application to a target device.

About Using the NativeUI Library

Use the NativeUI library to develop the user interface for a mobile application. It is recommended that you have your mobile application user interface design complete before starting to develop it using the NativeUI library.

Design Considerations

When designing the user interface:

- Review information about designing mobile applications. See "Mobile Application Design" on page 12.
- Understand the NativeUI object hierarchy. See "Hierarchy of NativeUI Objects for a User Interface" on page 13.
- If the application will run on a tablet device, you can design the user interface to use multiple panes. See "Using Multiple Panes for Tablet User Interfaces" on page 36.
- If you want the application to run on both tablet devices and smaller devices, you need to design for both. See "Designing Applications to Run on Both Tablets and Smaller Devices" on page 38.

Developing the User Interface Using the NativeUI Library

To create a user interface using the NativeUI library, use the Mobile Designer Java API, specifically the classes in the com.softwareag.mobile.runtime.nui package. The classes in this package control the NativeUI objects at the specific device level. Each NativeUI object maps to a platform-specific object for a target device, such as Apple's iPhone and Google's Android. Additionally, you can write your own extensions to the Mobile Designer NativeUI by extending any of the supplied classes or by creating new classes that add functionality.

See the following for information that is useful when coding the application:

- "Setting and Querying NativeUI Object Attributes" on page 14
- "Handling Events Generated by User Actions" on page 15
- "Transitioning Between Windows and Views" on page 16
- "Defining the Layout of Objects in the User Interface" on page 18
- "Managing Object Focus" on page 31
- "Background Colors and Images" on page 32
- "Adding Support for Right-to-Left Languages" on page 33
- "Using Multiple Panes for Tablet User Interfaces" on page 36

Mobile Application Design

Before coding a mobile application, you should design the application and determine its user interface.

Designing a Mobile Application

The first stage in a development process is design. Designing a mobile application involves defining the content, goals, and process flow of the application. During this stage, you should not be concerned with how the user will physically interact with the application. It is recommended that the outcome of the design phase is a thorough flow diagram that will enable you to effectively develop the application.

Designing the Mobile Application User Interface

After you design the application, you know the underlying processes that the application will need. You then design the user interface to determine how the user will interact with the application to achieve the application's intended functionality and goals.

When designing the user interface, determine the best way to present the required information to the user. A good approach is to start with a hub-and-spoke style application map that defines screens and the interactions with those screens to achieve

all steps within the application process flow. You can then draft the various individual screens using wireframe illustrations.

The Mobile Designer NativeUI library simplifies designing the user interface because it allows you to be less concerned with platform-specific differences of the target devices.

When designing a user interface for a mobile device, keep in mind:

- The size of a mobile device is much smaller than traditional, desktop applications. As a result, you will need to divide the information that you present to the user into multiple screens. Mobile applications tend to use a linear methodology, allowing the user to move from one screen to the next working on a single task at a time.
- Screen resolution can vary within platforms. As a result, deploying to a particular platform might mean building the application at more than one resolution. For example, Android devices running version 2.3 often had screen resolutions of 320x480 pixels. However, it is not uncommon for Android devices running version 4.0 to have resolutions of 720x1280 pixels. Using the NativeUI simplifies this because the NativeUI adapts to the varying screens sizes and resolutions, allowing text, buttons, and other user interface objects to render correctly. However, you still need to carefully consider the position and size of the objects you display in the user interface, including graphic images.
- High-resolution tablet devices require very large-size graphics for their high-density screens, that is more pixels per inch (PPI).
- Avoid designing a platform-specific user interface for an application that will run on many platforms. The unique interface mechanics of a platform might be impossible to render on other platforms. Additionally, users might find the unique interface mechanics hard to use because they are unfamiliar with them.
- Using the NativeUI library with little to no custom objects reduces issues, such as resolution dependency, localization, and accessibility.
- Be aware that user interface elements might not render the same way within a platform due to changes between versions of the platform. For example, the design of the Apple iOS On/Off switch changed visually between version 4.x and 5.x of the operating system. Also, the Apple iOS numeric keypad looks different on the iPad from the iPhone.

Hierarchy of NativeUI Objects for a User Interface

The NativeUI follows a strict hierarchy of visible components.

- **Windows** are at the top of the user interface hierarchy.

  A window defines the visible bounds of the NativeUI display. The application first displays a window. The application can then add views (that is, menus and screens) and other items related to the application inside the window. A window can:

  - Use the device's full display

- Use the device's full display excluding a status bar
- Be a dialog that uses only a portion of the device's display

An application might only require one window that the application uses to display each of the application screens as the user navigates between them. However, if the application requires window overlay, you can add multiple window support to the application.

Depending on the requirements of the window and the target platform, different additional features might be present. For example, on some platforms an overlay dialog might include a title bar that allows a user to reposition the dialog on device's display. Another example is that a title bar might contain a Close button.

If an application runs on a tablet device, you might want to use multiple panes within a window. You can then add views into the panes. For more information, see "Using Multiple Panes for Tablet User Interfaces" on page 36.

- **Views** are second level in the user interface hierarchy.

  The application displays a view within a window. Views are analogous to individual menus or screens in a user interface flow. A view can have a header bar, soft key labels, or encompass the entire window. Mobile Designer provides some custom views that automate the creation of common displays. For example, the nUINavView object is a view for navigation.

- **Elements** are the last level of the user interface hierarchy.

  Applications can add elements into views. Elements are singular display items or control items. Elements can have focus, and they can be selected. Examples of elements are images, buttons, and text fields.

  Elements can visually respond differently on different platforms and devices. For example, an edit field might present an overlay keyboard on one device to enter the data, while on another device, the edit field could provide text to speech functionality, or a handwriting recognition panel.

Setting and Querying NativeUI Object Attributes

The NativeUI objects have attributes associated with them. For example, an object might have a `Width` attribute or a `Height` attribute.

You can set attribute values in two ways:

- You can initially set an attribute value for an object by passing the value as part of the object's constructor when creating the object.
- After the object is created, you can change the value using a setter, for example setWidth().

> **Note:** You cannot change attribute values that are set in the constructor unless there is a corresponding setter for the attribute.

In addition to setting attribute values, once an attribute is created, you can query its value using a getter, for example getWidth().

Note: You might have to wait until the element is drawn on the screen before getting platform-level display metrics, such as the object's width, height, and X/Y coordinates. For example, some platform widgets might return misleading values for their height, such as 0 (zero), when the widget has not been rendered on the screen.

Handling Events Generated by User Actions

When a user interacts with the application, for example, pressing a button in the user interface, events can be generated.

About Listeners

You should set up the application so that it listens for events and takes appropriate measures to handle events. To listen for events, set up the application to implement the nUIEventListener class and register the classes as event listeners. As a result, the application receives events related to the currently active NativeUI object.

You can define listeners for individual NativeUI objects so that an object can have its own listener or an alternate listener. To do so, add the listener directly to the object using nUIObject.addEventListener().

Types of Events

The types of events for which an application can listen are defined in the nUIConstants class. For more information, see *webMethods Mobile Designer Java API Reference*.

An application can listen for:

- Events that the NativeUI system generates

  When a user interacts with a NativeUI object in an application's user interface, the NativeUI system generates an event. For example, an `EVT_GAIN_FOCUS` event is generated when an object gains focus. The events that the NativeUI system can generate are defined by com.softwareag.mobile.runtime.nui.nUIConstants.

- HTTP events

  The HTTP events are `EVT_TRIGGER_HTTP_SUCCESS` and `EVT_TRIGGER_HTTP_FAIL`. These events are not related to any specific Mobile Designer classes. The HTTP events are available if the application requires this functionality.

- Custom-defined events

  You can define custom events. `CUSTOM_EVENT_CODE0` is the first constant value that is not reserved for use within the NativeUI system. When defining custom events, you can assign constant values that are equal to or greater than this value.

Sample Code that Manages Event Handling

The following code sample shows how the Mobile Designer NativeUIHelloWorld sample application manages event handling:

```
//see MyCanvas.java
public boolean nUIEventCallback(nUIObject object, int evt_type)
{
     switch (object.nuiid)
     {
     case NUIID_START_PROGRESS:
        if(evt_type == EVT_TRIGGER)
           transitionToView (main_view, onCreateEndView());
        break;
     case NUIID_END_BACK:
        if(evt_type == EVT_TRIGGER)
           transitionToView (main_view, onCreateStartView());
        break;
     }
     return true;
}
```

Setting a Unique Identifier for NativeUI Objects So that You Can Identify Them When Listening for Events

When an event occurs, the NativeUI system passes the NativeUI object that generated the event and the event type to the event listener. Each NativeUI object has a unique identifier. This unique identifier is the *nuiid* value that the application passed to the constructor when creating the NativeUI object. In this example that uses `NUIID_START_PROGRESS`, the following code shows the `NUIID_START_PROGRESS` unique identifier:

```
//Specify the ID for the start progress button.
//Use any number as long as it is unique.
public static final int NUIID_START_PROGRESS = 0x01020101;

//After specifying the ID, in onCreateStartView()
start_view.add(new nUIButtonElement(NUIID_START_PROGRESS, "Progress");
```

Tip: If you are not concerned about events for a NativeUI object, use an appropriate constructor without a *nuiid* value.

Return Values from Event Processing

After an application handles an event, it should return `true` or `false` to indicate whether the NativeUI system should perform the default behavior for the event. In most cases, the application should return `true` to indicate that the NativeUI system should perform its default behavior for the event.

Transitioning Between Windows and Views

An application initially displays a window. Once a window is displayed, to display a view within a window or display another window, the application needs to perform a

transition to the new location. Mobile Development handles this automatically. But it can also be coded using the following methods and classes.

- To transition to a new view within a window, use the transitionTo and transitionFrom methods in the nUIWindowDisplay class.
- To transition to a new window, use the nUIController class.

For more information about the nUIWindowDisplay and nUIController classes, see *webMethods Mobile Designer Java API Reference*.

The following code sample is a portion of the code from the Mobile Designer NativeUIHelloWorld sample application. It shows how to use the transitionTo and transitionFrom methods in an application:

```
private void transitionToView(nUIViewDisplay new_view, int pane)
{
   int transition = nUIController.TRANSITION_APPEAR;
   nUIViewDisplay old_view = main_view;
   if (old_view != null)
   {
       if(old_view.nuiid < new_view.nuiid)
          transition = nUIController.TRANSITION_LEFT;
       else if(old_view.nuiid > new_view.nuiid)
          transition = nUIController.TRANSITION_RIGHT;
       main_window.transitionFrom(old_view,transition, pane);
   }
   main_window.add(new_view);
   main_window.transitionTo(new_view, transition, pane);
   main_view = new_view;
}
```

The code sample illustrates how to replace a window's current view with a new one by:

1. Using the transitionFrom method to transition away from the current view.
2. Using the add method to add the new view to the window.
3. Using the transitionTo method to transition to the newly added view.

The code sample uses a view's unique identifier to determine the transition direction (either `TRANSITION_LEFT` or `TRANSITION_RIGHT`). If the new view has a lower unique identifier, the code transitions one way. If the new view has a higher unique identifier, it transitions the other way. This transition logic represents only one approach. There are other transition logic approaches that you can use to meet the requirements of your mobile application.

The NativeUI systems supports the following transition properties that are defined in the com.softwareag.mobile.runtime.nui.nUIController class:

- `TRANSITION_APPEAR`
- `TRANSITION_FADE`
- `TRANSITION_LEFT`
- `TRANSITION_RIGHT`
- `TRANSITION_UP`
- `TRANSITION_DOWN`

All platforms support the `TRANSITION_APPEAR` property. However, platforms might substitute alternative solutions for the other nUIController class transition properties.

Defining the Layout of Objects in the User Interface

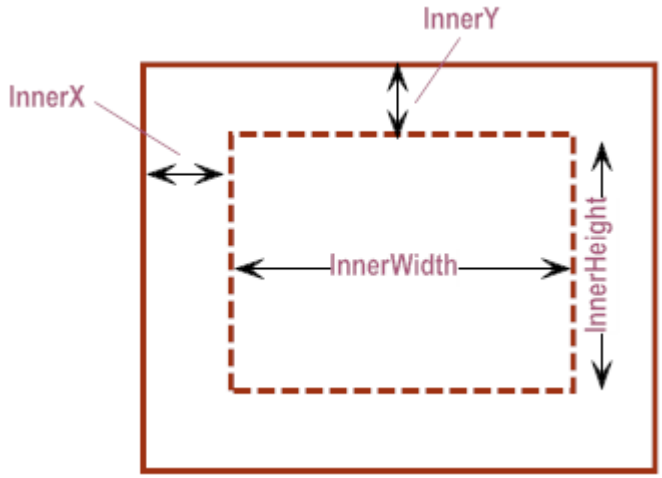
To create a user interface, code the application to first add a window object. Inside a window, place a view. The application can then add additional NativeUI objects into the view. For more information, see "Hierarchy of NativeUI Objects for a User Interface" on page 13.

NativeUI objects can be thought of as parent objects and elements. *Parent objects* contain other NativeUI objects, which are referred to as *elements*. Examples of parent objects are views, scrollable containers, and table cells. Examples of elements are text entry fields, buttons, and images.

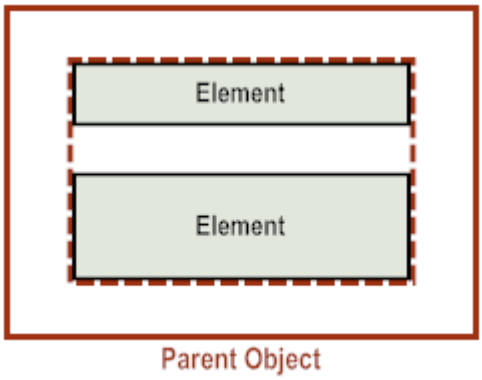
Controlling the Inner Padding of Parent Objects

Parent objects have *inner padding*. If an object is displayable, that is an object that inherits from nUIDisplayObject, and is also an object in which you can insert child elements, you can control a parent object's inner padding using the following attributes of the parent object:

Inner Padding Attribute	Description
`InnerX`	Defines the distance from the parent object's left edge to where child elements are drawn.
`InnerY`	Defines the distance from the parent object's top edge to where child elements are drawn.
`InnerWidth`	Defines the parent object's usable width in which you can add content.
`InnerHeight`	Defines the parent object's usable height in which you can add content.



The parent object's inner padding causes the child elements to be indented from the edges of the parent. This is a useful concept to take advantage of when you do not want items to touch the edges of screens or borders.



The NativeUI objects have default inner padding values. At the nUIDisplayObject level, all the attribute values are set to `0` (zero). However, the attribute values are overridden for some displayable NativeUI objects to match the expectations for each platform. Specifically, the objects for tables, views, and scrollable containers might override the default values. When using NativeUI objects for an application, you can override the default inner padding attribute values to meet the needs of your application.

If you set the attribute value for `InnerX`, but not `InnerWidth`, by default the `InnerWidth` value is determined by mirroring the `InnerX` padding on the other side.

```
InnerWidth = overall_width_available - (2 * InnerX)
```

Similarly, if you set the attribute value for `InnerY`, but not `InnerHeight`, by default the `InnerHeight` value is determined by mirroring the `InnerY` padding on the bottom edge.

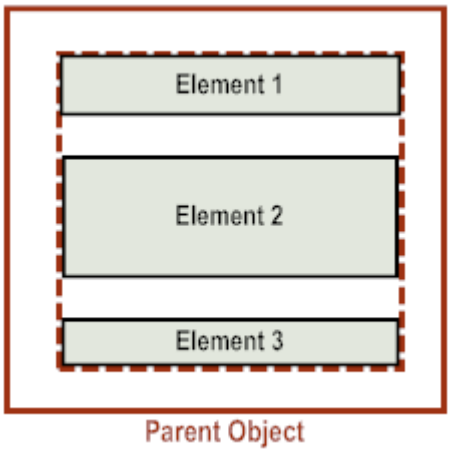
```
InnerHeight = overall_height_available - (2 * InnerY)
```

If you do not want to use this default behavior, you can explicitly set the `InnerWidth` and `InnerHeight` attributes for an object.

Note: If you explicitly set the `InnerWidth` and `InnerHeight` attributes to set the width and height of the parent object, your application logic will also have to handle any size adjustments due to the resizing of the parent object or changes to orientation of the device.

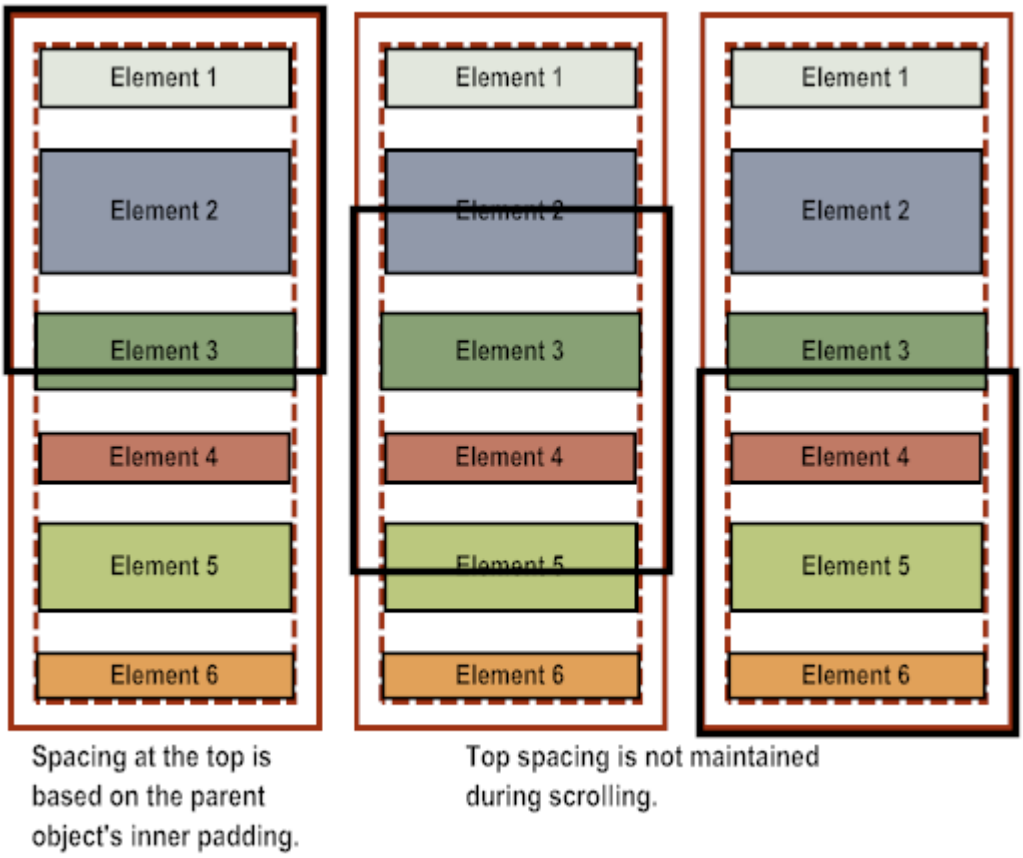
Positioning Elements in a Parent Object

By default, when an application adds elements to a parent object, the elements are positioned vertically, one below the other, starting at the top of the parent object.

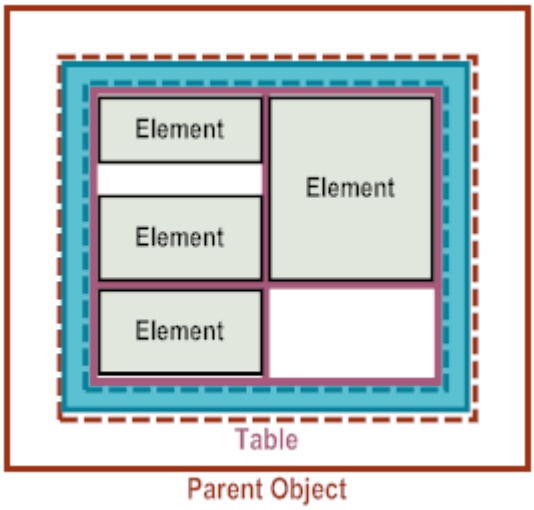


The elements are indented based on the inner padding of the parent object. When originally displayed, the first element is spaced from the top of the parent object based on the vertical inner padding. For more information, see "Controlling the Inner Padding of Parent Objects" on page 18.

If you place elements in a scrollable parent object, for example a view or scrollable container, the top padding is not maintained when a user scrolls through the contents.



If you want to position child elements side by side within a parent object, use a table and place elements within table cells. For more information, see "Using Tables to Control the Layout of Elements" on page 26.



As an alternative to using the default layout or positioning elements using a table, you can use absolute positioning. To do so, set the `X`, `Y`, and `Width` attributes of the child elements that you add to the parent object. While absolute positioning gives you the

greatest amount of control for exact positioning, using absolute positioning prevents the application's user interface from automatically adapting to:

- Different size devices
- Different size of user interface elements among the various platforms
- Re-aligning user interface elements when the orientation of the device is changed

If you use absolute positioning, you must add logic to your application to handle these types of issues.

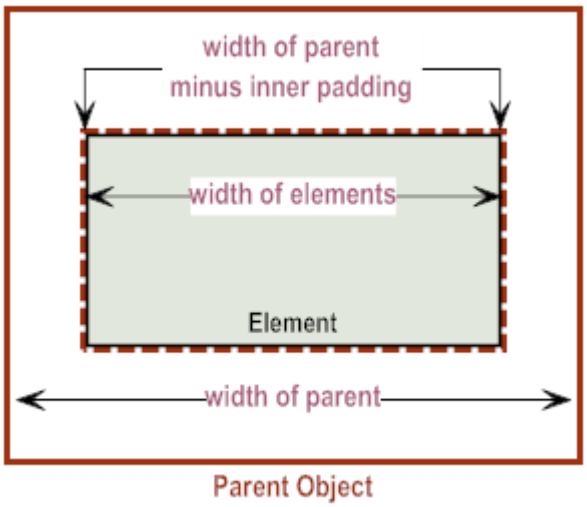
Sizing Child Elements

The height of a child element is determined by the data for the element.

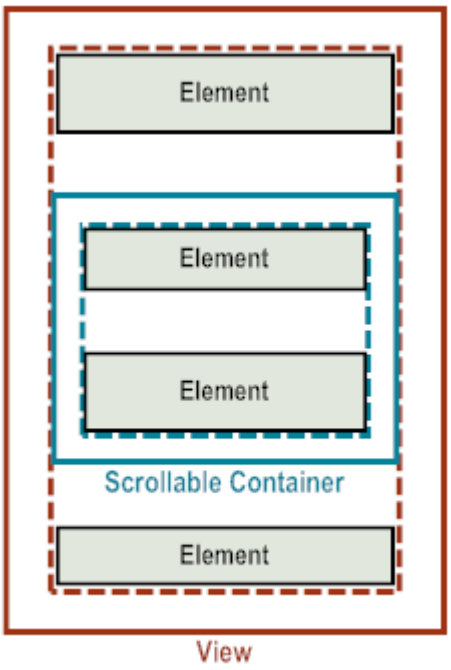
Note: For most elements (buttons, text, images, etc.), using the `Height` attribute to explicitly set the number of pixels for the element's height is not recommended.

For the width of an element, you can use the default, or you can set the child element's `Width` attribute to explicitly specify the number of pixels to use for the element's width.

The default element width is the width of its parent object minus the inner padding. For more information about a parent object's inner padding, see "Controlling the Inner Padding of Parent Objects" on page 18.

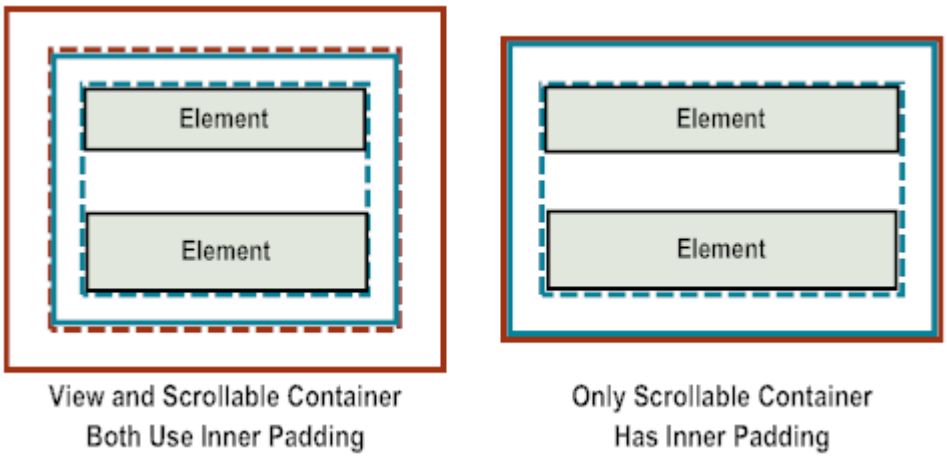


If you nest parent objects, child elements placed in the inner parent object are narrower because the inner padding values are compounded. For example, you might nest a scrollable container inside a view. Elements added to the view have the width of the view's inner width. However, elements added to the scrollable container are narrower due to the inner padding of both the view and the scrollable container.



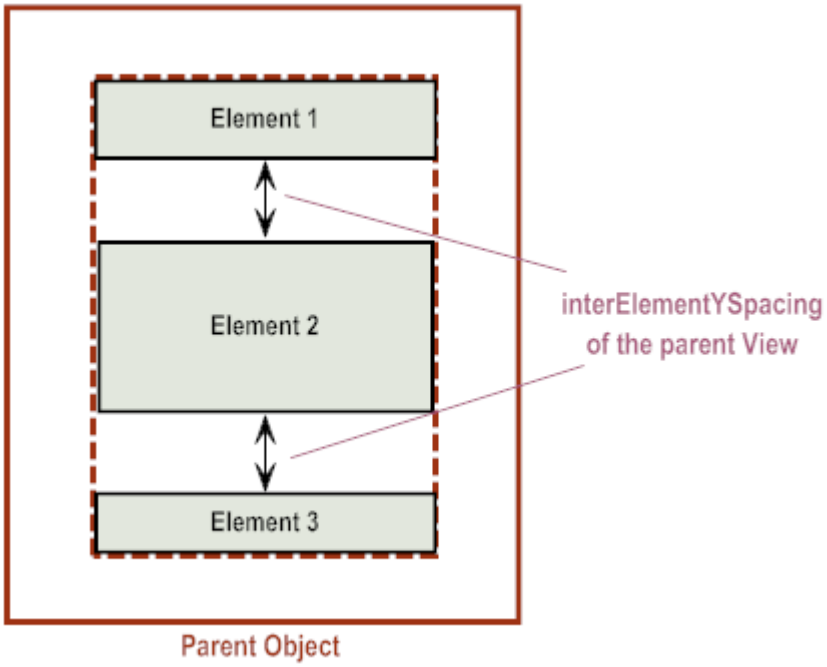
When nesting NativeUI objects, you might want to adjust the inner padding of objects. For example, in the example of a scrollable container inside the view, if you set the scrollable container's `InnerX` to `0` (zero), the widths of the elements both in the view and the scrollable container will be the same.

In other instances, you might want to remove the inner padding from the outer NativeUI object. For example, you might have a view that contains a scrollable container, but no other child elements. The scrollable container might contain additional elements. In this situation, the inner padding of the view compounded with the inner padding of the scrollable container results in wasted screen space. As a result, you might want to set the view's `InnerX` and `InnerY` values to `0` to remove excess padding.



Controlling the Vertical Spacing Between Child Elements

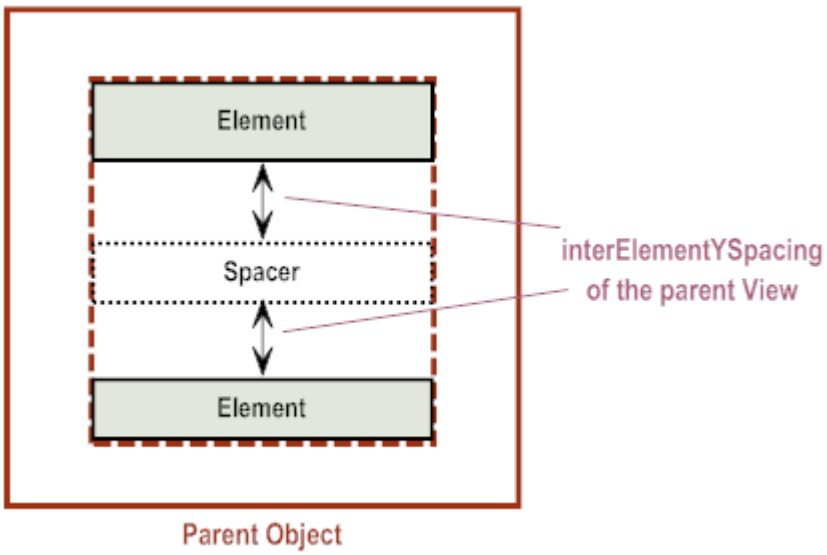
By default, the space between the elements is determined by the `InterElementYSpacing` attribute of the parent view. The application can alter the space between the elements by specifying a pixel value for the parent view's `InterElementYSpacing` attribute.



If you want additional space between two elements, use the following NativeUI objects:

- Insert the nUISpacerElement NativeUI object to add additional white space between two elements. Use the nUISpacerElement NativeUI object's `Height` attribute to specify the pixel height of the white space.
- Insert the nUISeparatorElement NativeUI object to display a horizontal line between two elements. Use the nUISeparatorElement NativeUI object's `Height` attribute to specify the pixel height of the nUISeparatorElement object. The horizontal line displays in the vertical center of the object.

When determining the vertical height you want to use for the nUISpacerElement and nUISeparatorElement objects, take into consideration that the parent view's `interElementYSpacing` also displays around the object.



Note: When using grouped buttons on platforms where grouped buttons are supported (primarily iOS), buttons in the same group will not have vertical space between them, regardless of the value of the parent view's `InterElementYSpacing` attribute.

Controlling the Horizontal Alignment of Elements

You can control the horizontal alignment of some elements. If a NativeUI object has a setHalign method, for example, nUITextfieldElement object, you can use the method to control the object's horizontal alignment to align its contents left, center, or right within the parent object. Similarly, other NativeUI objects might use a setAlign method, such as the nUIImageElement object.

Even if visually, an element does not use the full screen width, as usual, subsequent elements are added one below the other. For example, consider an example where you have two text elements, where the first is left-aligned and the second is right-aligned. Visually, the text in the elements might not span the entire width of the screen.

Using Tables to Control the Layout of Elements

If you want a more complex layout than just elements positioned vertically, one below the other, you can use tables for your layout. When you use tables, you can:

- Use background colors or images to change the background of the entire table, entire rows, and/or individual cells.
- Add borders around cells and control the thickness of the cell borders.
- Position elements side by side by placing the elements in table cells.
- Span cells horizontally and or vertically.

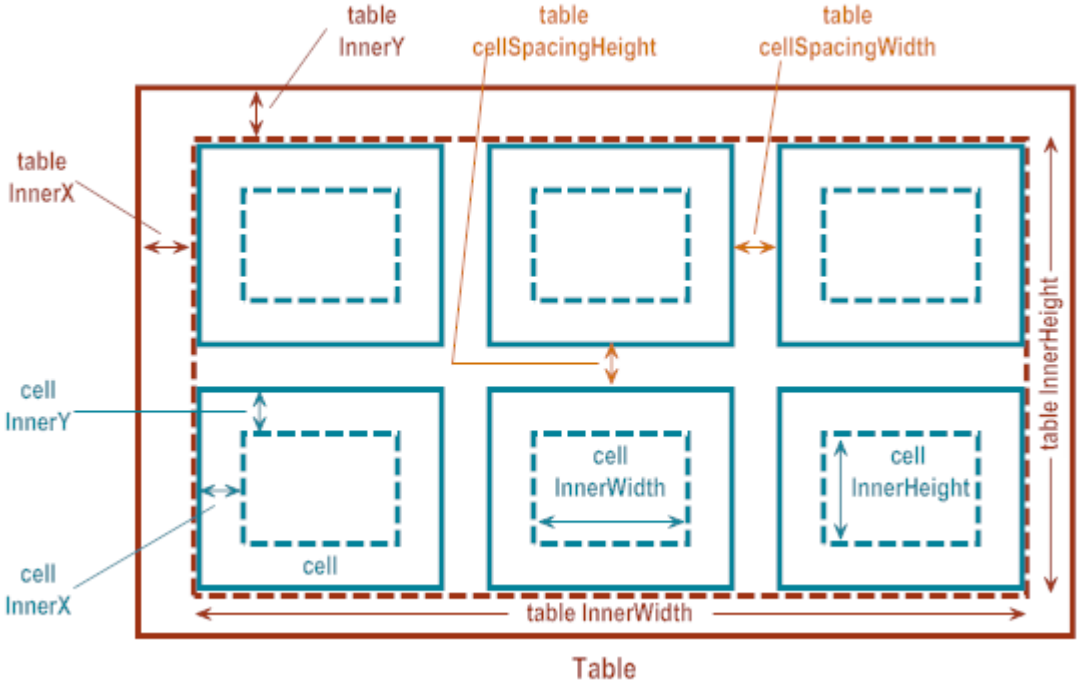
Using tables allows you to precisely position elements while still allowing your user interface to scale to all devices, platforms, font sizes, and orientations.

Controlling Inner Padding and Spacing in Tables

When using tables (nUITableElement objects) for element layout, you must consider the table's inner padding and the cell spacing within a table. Additionally, table cells (nUITablecellElement objects) also have inner padding. You control inner padding and spacing using the following attributes.

NativeUI Object	Attribute	Description
Table (nUITableElement object)	`InnerX`	Defines the distance from the table's left edge to where table cells are drawn.
	`InnerY`	Defines the distance from the table's top edge to where table cells are drawn.
	`InnerWidth`	Defines the table's usable width in which you can add content.
	`InnerHeight`	Defines the table's usable height in which you can add content.
	`cellSpacingWidth`	Defines the distance between the table columns.
	`cellSpacingHeight`	Defines the distance between table rows.

NativeUI Object	Attribute	Description
Table cell (nUITablecellElement object)	`InnerX`	Defines the distance from a cell's left edge to where child elements are drawn.
	`InnerY`	Defines the distance from a cell's left edge to where child elements are drawn.
	`InnerWidth`	Defines a cell's usable width in which you can add content.
	`InnerHeight`	Defines a cell's usable height in which you can add content.



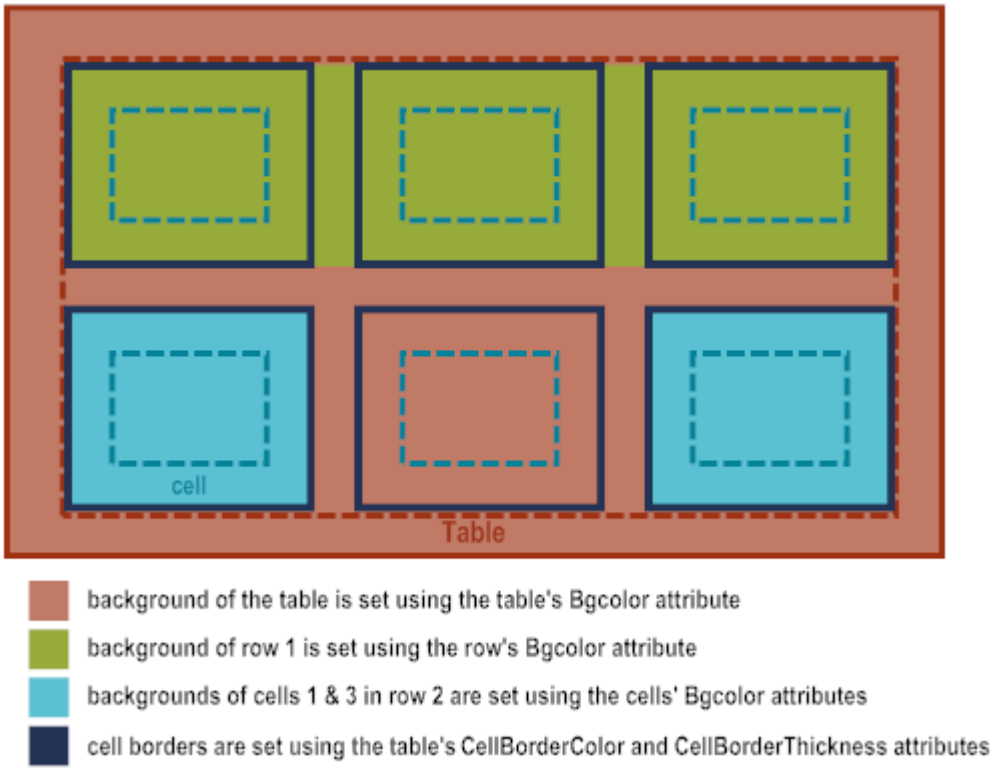
The use of `InnerX`, `InnerY`, `InnerWidth`, and `InnerHeight` attributes in tables and table cells is the same as for any parent object. For more information, see "Controlling the Inner Padding of Parent Objects" on page 18.

Adding Background Colors, Images and Borders

You can set the background color and image of tables (nUITableElement objects), table rows, (nUITablerowElement objects), and table cells (nUITablecellElement objects). Additionally, you can add borders around the cells in a table. You control background colors, images, and borders using the following attributes.

NativeUI Object	Attribute	Description
Table (nUITableElement object)	`Bgcolor`	Defines the background color for the entire table.
	`BackgroundDrawable`	Defines the background color or image for the entire table.
	`CellBorderColor`	Defines the color for the borders drawn around the cells in the table.
	`CellBorderThickness`	Defines the width of the borders drawn around the cells in the table. Specify a pixel value for the width. If you do not want the cells to have a border, specify `0` (zero).
Table row (nUITablerowElement object)	`Bgcolor`	Defines the background color for an entire row.
	`BackgroundDrawable`	Defines the background color or image for an entire row.
Table cell (nUITablecellElement object)	`Bgcolor`	Defines the background color for a cell.
	`BackgroundDrawable`	Defines the background color or image for a cell.

The following illustrates how to use the attributes for background color and cell borders.



In the illustration above:

- For row 1, the row background color is set to green. Because no background colors are set for the cells in row 1, the row background color displays in the cells.
- For row 2, the background color for the row is not set. As a result, the table background color displays for the row. Cells 1 and 3 in row 2 have a background color set to blue. The background color is not set for the cell 2 in row 2, so it takes on the background color of the table.
- The table's cell borders are set to dark blue. As a result, all cells in the table have a dark blue border.

Sizing Table Columns, Rows, Cells, and the Elements Placed in Cells

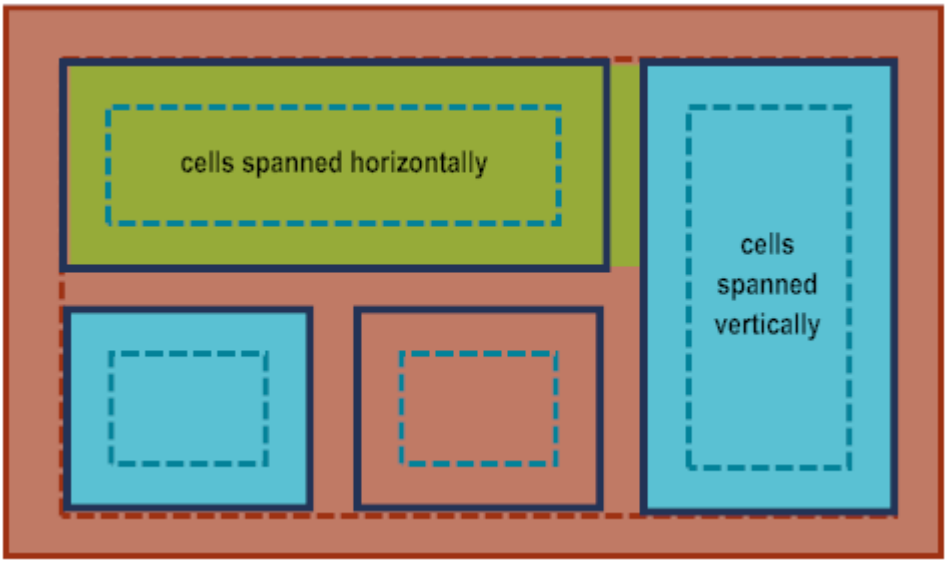
When you add a table (nUITableElement object) to a parent object, the table uses the full width available in the parent object. The following describes how columns, rows, cells, and elements are sized.

- **For columns**, you specify the relative column width sizes when using the constructor to create a nUITableElement object. For example, if you specify `1, 2, 1`, the constructor creates a table with 3 columns where column 1 and 3 are half the size of column 2. In other words, column 1 uses 25% of the table width, column 2 uses 50% of the table width, and column 3 uses the remaining 25%.
- **For table rows** (nUITablerowElement objects), by default, the height of a row is determined by the height of the cells that the row contains. If you want, you can set the `Height` attribute of a nUITablerowElement object to specify a pixel value to use for the row

height. However, if the cells in the table are larger than the pixel value you specify, the content is clipped.

- **For cells** (nUITablecellElement objects):
  - Cell width is determined by the size of the column in which the cell resides, and also taking into consideration the column spacing, which is the gap between columns in the table. The column spacing is set using the `CellSpacingWidth` attribute of the nUITableElement object.
  - Cell height, by default, is determined by the height of the cell's contents and inner padding. If you want, you can set the `Height` attribute of a nUITablecellElement object to specify a pixel value to use for the cell height. However, if the contents of a cell is larger than the pixel value you specify, the content is clipped.

    You can span table cells both horizontally or vertically. To span cells horizontally, use the `Hspan` attribute of the nUITablecellElement object. To span cells vertically, use the `Vspan` attribute of the nUITablecellElement object.



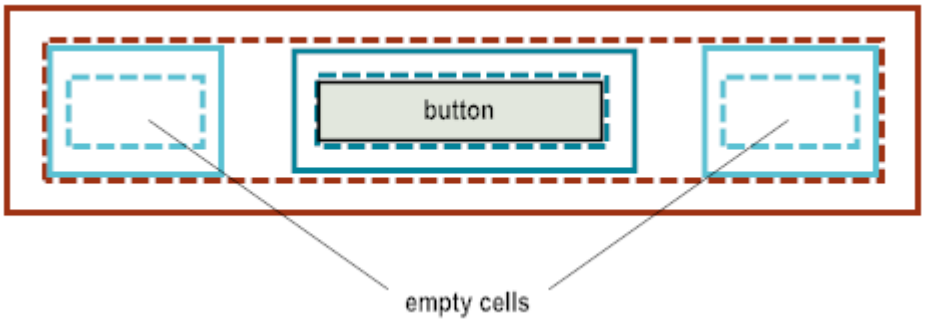
- **For elements** that you place in table cells, the element width and height is determined in the same way as placing the elements in any parent object. For more information, see "Sizing Child Elements" on page 22.

Positioning Elements in Table Cells

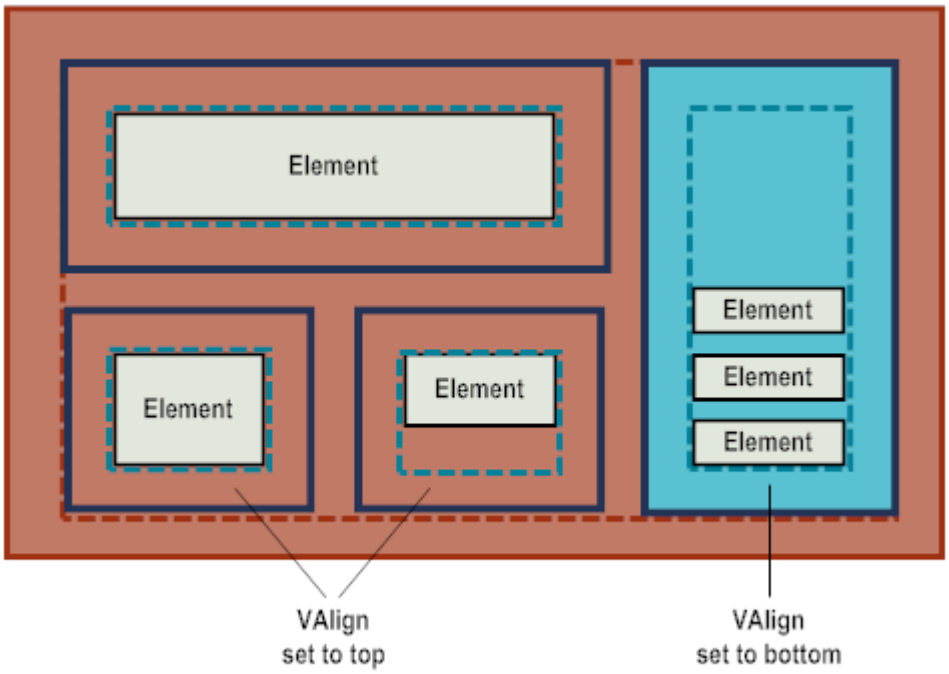
When an application adds elements into a table cell, the elements are positioned vertically, one below the other, starting at the top of the table cell. The elements are indented based on the table cell's inner padding. You can control the spacing between elements in a table cell in the same way as you control vertical spacing for any parent object. For more information, see "Controlling the Vertical Spacing Between Child Elements" on page 24.

You can leave cells empty. For example, an application might use a single row table with column widths set to 25%, 50%, 25% of the table width. To have a button that is 50% the

size of the table width display in the center of the screen, the application can place the button in the center cell, leaving the outer cells empty.



You can use the `VAlign` attribute of the nUITablecellElement object to vertically align the contents of a cell. This is useful when tables contain cells that are vertically spanned, and also when the elements in table cells can potentially be of different heights (images, text, buttons, etc.). The `VAlign` attribute allows applications to vertically align elements in a manner that looks good on the device.



Managing Object Focus

In an application's user interface, when a NativeUI object gains focus, its appearance changes to indicate that it is ready for user interaction, such as to receive input from a finger tap or keyboard. How the appearance of a NativeUI object changes depends on the platform. Platforms use different visual clues, such as highlighting the object, making the object visually distinct, or changing the color of the background behind the object.

The NativeUI system has default behavior for whether a newly added object gains focus. By default, when adding a view to a window or a focusable NativeUI object to a view, the following behavior occurs:

- If the parent object does not already contain an object that has focus, the newly added object gains focus.
- If the parent contains an object that has focus, the focus does not change when the new object is added to the view or window.

You can override the default behavior for NativeUI objects that are a subclass of the nUIDisplayObject class by using the parent.setChildFocus(child_to_focus_on) method.

Background Colors and Images

The background of most NativeUI elements can be changed using two properties, `Bgcolor` and `BackgroundDrawable`. Setting a value for `Bgcolor` will override any previously set `BackgroundDrawable`, and similarly, setting a `BackgroundDrawable` will override any previous `Bgcolor`.

`Bgcolor` is the older of the two properties and has been extended in Mobile Designer 9.12 to support more NativeUI elements (as platform support allows). As the name suggests, `Bgcolor` can only influence background colors. Setting `Bgcolor's` value to `nUIDisplayObject.COLOR_BACKGROUND_NORMAL (0xFFFE00FF)` is considered as a special value and corresponds to whatever the platform would normally do for this element, so it cannot be set directly as a literal color value.

`BackgroundDrawable` is a new property added to Mobile Designer 9.12. Using `BackgroundDrawable`, you can get a much wider set of options for a NativeUI object. A new package, com.softwareag.mobile.runtime.nui.background, contains the three types of background classes that can be set:

- ColorBackground for raw color values (including the literal value of `COLOR_BACKGROUND_NORMAL`, if required)
- PatternImage for images (with optional tiling or scaling if memory and platform support allows)
- DefaultBackground for the default background behavior of this element. Use the static reference `DefaultBackground.DEFAULT` if you want to reset a NativeUI object to it's default background.

Wherever possible, consider using the `BackgroundDrawable` property in preference to `Bgcolor`.

Adding Support for Right-to-Left Languages

The right-to-left (RTL) or left-to-right (LTR) property of a writing system is commonly referred to as its *directionality*. The NativeUI library has methods to support locales that require right-to-left directionality, such as Hebrew and Arabic.

You can change the directionality of the entire application's user interface to use right-to-left directionality. However, if necessary, an application can use a mix of right-to-left and left-to-right directionality.

Based on the platform on which an application is running, using right-to-left directionality for a user interface might change:

- Default alignment of the NativeUI objects and the text within the objects
- Position of the Back buttons and header menus within the user interface
- Ordering of the columns within the nUITableElement NativeUI object
- Positioning of the carat within the nUIEntryElement NativeUI object

Controlling the Directionality of an Application

The following table describes the NativeUI classes and methods you use to control directionality of an application:

NativeUI Class	Description
nUIConstants	Use the following integers to represent the directionality: ■ `TEXT_DIRECTION_LTR` for left-to-right directionality ■ `TEXT_DIRECTION_RTL` for right-to-left directionality
nUIController	Use the following methods to control the global behavior of the application: ■ deviceSupportsAppDirectionality() method The deviceSupportsAppDirectionality() method indicates whether the platform supports a specified directionality at run time. The method returns `true` if the platform supports the directionality or `false` if it does not. The method might return `false` for one of the following reasons: ■ Directionality support for the platform was unavailable through the NativeUI system at the current time. ■ The device's locale settings do not allow a directionality change at the current time.

NativeUI Class	Description

- The platform does not support the directionality due to other platform-specific issues.

- void setAppDirectionality(int direction) method

  Use the void setAppDirectionality(int direction) method to set the directionality. For direction, specify either `TEXT_DIRECTION_RTL` for right-to-left or `TEXT_DIRECTION_LTR` for left-to-right.

  After changing the directionality, the getAppDirectionality() method immediately reflects the new direction.

  During the next UI update, the class that extends MDApplication will be notified with the new direction.

  Note: You can override the MDApplication.appDirectionalityChanged() in your MDApplication class if you need to handle changes in the application directionality. Because your initial Canvas class already extends CanvasNativeUI, you do not have to use a separate class to override this function.

- getAppDirectionality() method

  The getAppDirectionality() method returns the application's current global directionality setting, either `TEXT_DIRECTION_LTR` or `TEXT_DIRECTION_RTL`.

nUITableElement

Use the following methods to manage the directionality within a nUITableElement NativeUI object:

- Use the setIgnoreDirectionality() method to have the NativeUI system ignore the application's current directionality setting.

  If an application's directionality is set to right-to-left, by default, the table's column order is reversed. If you do not want the columns reversed, use this method to ignore the directionality setting for the table columns. For example, you might want this if the table contains images that you want displayed in a specific order regardless of the application's directionality.

- Use the getIgnoreDirectionality() method to determine whether the application directionality will affect the ordering of columns within the table. The method returns `true` if column ordering will be affected by the application directionality or `false` if column ordering will not be affected.

Exceptions to Right-to-Left Directionality

The following lists situations when the application's directionality is not enforced:

- If an application explicitly sets the position of a NativeUI object, for example, using the setX method, the NativeUI system does not override the position. Similarly, if an application explicitly sets the alignment of a NativeUI object, for example, using the setHAlign method, the NativeUI system does not override the alignment. Only NativeUI objects that have the default position and alignment are subject to directionality changes.
- The nUIWebviewElement and nUIWebView NativeUI objects are not subject to the application's directionality. To change the directionality of text within these NativeUI objects, use the HTML `DIR` attribute, for example, `<HTML DIR="RTL">` or `<p DIR="RTL">`.
- The charting APIs are not subject to the directionality of an application. In most platforms, right-to-left text displays using a right-to-left direction. However, chart and axis reordering is not performed.

Platform-Specific Notes and Issues

When an application uses a right-to-left directionality, how objects in the user interface display depends on the platform's support for right-to-left setups. Wherever possible, the NativeUI system attempts to use the platform-specific conventions for when a device uses a right-to-left locale.

Platform	Notes
Android	■ Android version 11 and higher support right-to-left directionality. ■ The nUICheckboxButton, nUIRadioCheckbox, and nUIDropdownlistEntry NativeUI objects display using left-to-right directionality. Additionally, non-custom dialog boxes, that is, those created using the nUIAlertDialog NativeUI object, might also retain left-to-right alignments.

iOS	■ The following NativeUI objects display using left-to-right directionality.
	  ■ nUICheckboxButton
	  ■ nUIDropdownlistEntry
	  ■ nUIRadioCheckbox
	  ■ nUINavView
	■ The buttons within a nUIDialogWindow NativeUI object display using a left-to-right directionality.

Platform	Notes
J2ME (Phoney)	■ The NativeUI system attempts to provide directionality features similar to those that other platforms provide. In general, the NativeUI system makes the functionality align with that provided by Android and iOS.

Using Multiple Panes for Tablet User Interfaces

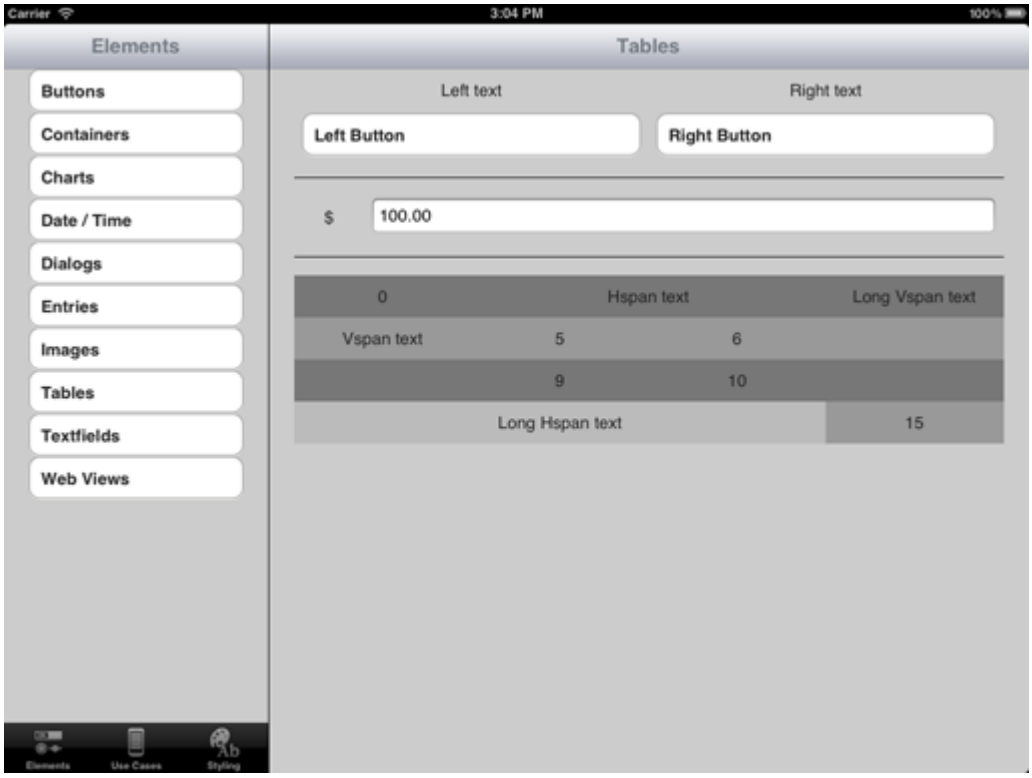
The NativeUI window object, nUIWindowDisplay, allows you to define multiple panes in a window. Using multiple panes in a window is primarily useful when creating user interfaces for tablet devices that have larger screen sizes.

By default, the nUIWindowDisplay object has two panes, one with a nUIViewDisplay object for the main pane and a nUINavView object for navigation. You can configure the nUIWindowDisplay object to accept additional nUIViewDisplay and nUINavView objects, allowing the application to use multiple panes to take advantage of the larger screen size.

Important: Before creating an application that uses multiple panes, ensure the target platforms on which the application will run support windows with multiple panes.

NativeUIDemo Sample

The Mobile Designer NativeUIDemo sample application is an example of an application that uses multiple panes. It is also an example of an application that runs on both tablet devices and smaller devices, altering the user interface based on the device size. The following shows a screenshot of the NativeUIDemo.



If the NativeUIDemo application determines that it is running on a tablet device, it uses multiple panes. To determine whether it is running on a tablet, the application derives the screen size of the device on which it is running. The NativeUIDemo sample considers a tablet to be a device with a screen size that exceeds six inches diagonally. For more information, see "Determining the Device Size at Run Time" on page 38.

Managing the Layout of Panes

The common layout for tablet applications in landscape mode is to divide the screen into two panes with a smaller, navigation pane and a larger, main pane.

When using the NativeUI, you have complete control over the layout of the panes. By default, the nUIWindowDisplay object has a navigation pane and a main pane. The main pane occupies all space that is not used by the navigation pane. You can add additional panes and divide the window into as many panes as you need.

You should carefully plan how to arrange and size the panes. Consider how the panes will work on screens with differing resolutions and how device orientation will affect the usability of the application.

Important: It is recommended that you do not use overlapping panes.

You can add logic to your application that determines the size of the panes at run time. For example, the NativeUIDemo sample uses a flexible method to determine the size of its panes. The application sets the width of the left pane to the smaller of either 40% of the overall screen width or 2 inches. As a result, the user interface is usable even if

the application is running on a small tablet in portrait mode. The application sets the height of the left pane to the full screen height less the height of the navigation bar. The right, main pane fills the remaining space. For more information, see "Adding Panes to a Window" on page 39.

Designing Applications to Run on Both Tablets and Smaller Devices

You can create applications that support both multiple panes aimed for larger devices, such as tablets, and single screens aimed for smaller devices, such as smartphones. When designing the application, it is recommended that you initially design the application flow for the multiple-pane version as a set of features that you can degrade gracefully to accommodate the single-screen version. Alternatively, you can design the single-screen version, and after the application is complete, convert it to a multiple-pane application.

When designing the application flow, be sure to consider the differences between displaying information using multiple panes vs. a single pane. For example, the flow for a single-screen application tends to be linear. Because the logic is more linear, the behavior of the Back button is somewhat predictable. When designing the application flow for an application that will use multiple panes, you can divide the tasks between the panes. Action in one pane can trigger changes in other panes. Because of the possibility of changes being triggered in separate panes, in a multiple-pane application, it is less obvious how and when to display the Back button.

Another example of a difference is the navigation bar. For a single-screen application that is aimed for smaller devices with limited screen size, you might need to limit the icons displayed on the navigation bar. When using multiple panes for a larger device, you have more room to display icons. Additionally, you can split the contents of the navigation bar across multiple panes.

When creating an application that supports both multiple-pane and single-screen versions, you need to add logic to determine when to use the logic for the multiple-pane version or the single-screen version. One method is to base the decision by determining the size of the device. For more information, see "Determining the Device Size at Run Time" on page 38.

Determining the Device Size at Run Time

If a target platform supports multiple panes, one way to determine whether to use the multiple-pane logic rather than use a single-screen logic is by determining the size of the device on which the application is running. After determining the size of the device, the application can then execute the appropriate logic for the device size.

At run time, the physical screen size of the device is not available because this value is not stored anywhere. However, the application can determine the screen size by checking the screen resolution against the screen's pixels per inch (PPI). The screen resolution and PPI values are stored in the device profile. At run time, the application can query the values using the nUIController.getScreenWidth(), nUIController.getScreenHeight() and nUIController.getScreenPPI() methods.

You code the application to determine what size is considered a tablet. For example, the logic might consider that any device with a screen size that exceeds six inches diagonally is a tablet.

To see sample code that performs this logic, review the code in the Mobile Designer NativeUIDemo sample.

Adding Panes to a Window

By default, the NativeUI nUIWindowDisplay object has two panes, Pane 0 and Pane 1. Pane 1 is for navigation using the nUINavView object. Pane 0 is the main pane and occupies all space that Pane 1 does not use.

The code examples in this section show how to add an additional third pane to the left side of a window.

The following code example is for a setPaneDimensions method. It defines the dimensions for three panes: the main pane, the navigation pane, and the additional side pane. The code explicitly defines the dimensions of the side pane and the navigation pane. The side pane occupies 40% of the total screen width or two inches, whichever is the smaller. The pane for the navigation bar uses the full width of the screen. The main pane occupies the remaining available space.

```
int mainpane = 0;
int navpane = 1;
int sidepane = 2;
nUIWindowDisplay main_window;
nUINavView main_navbar_view;
protected void setPaneDimensions()
{
 int sidepane_width = Math.min ((CURRENT_SCREEN_WIDTH * 40) / 100,
CURRENT_SCREEN_PPI * 2);
 int navbar_height = 0;
 if (main_navbar_view != null)
//the navigation bar is not used everywhere in the application
 {
  navbar_height = main_navbar_view.getHeight ();
 }
 int height = main_window.getHeight ();
 main_window.setPaneDimensions (sidepane, new int [] { 0, 0, sidepane_width,
height - navbar_height });
 main_window.setPaneDimensions (mainpane, new int [] { sidepane_width, 0,
CURRENT_SCREEN_WIDTH - sidepane_width, height - navbar_height });
//navigation pane is full-width and calculated automatically.
}
```

After defining the setPaneDimensions method, it can be invoked during onCreateMainWindow when creating the main window of the application. By doing so, the setPaneDimensions method creates the pane structure. You should define the pane structure as soon as the screen dimensions and screen pixels per inch (PPI) are available.

The following code example shows how to create the main pane, navigation pane, and side pane, as well as showing how to set transitions. Note that the logic adds the navigation pane before setting the pane dimensions so that the setPaneDimensions method can adjust the height.

```
nUIViewDisplay main_view, side_view;
//onCreateMainWindow is called from CanvasNativeUI.
protected nUIWindowDisplay onCreateMainWindow()
{
 main_window = new nUIWindowDisplay(NUIID_MAIN_WINDOW);
 main_window.add(onCreateMainNavbarView());
 main_view = onCreateMainView();
 side_view = onCreateSideView();
 setPaneDimensions(); //size panes according to contents
 transitionToView(main_view, mainpane);
 transitionToView(side_view, sidepane);
 return main_window;
}
```

Note: By default, the NativeUI system assumes that the application uses two panes, Pane 0 and Pane 1, and uses the default size for each. If an application uses additional panes, the NativeUI system must be aware of the additional panes. To do so, in onCreateMainWindow, the application logic should call the setPaneDimensions method before adding content to panes higher than 1. In the above example, that means before adding content to the additional side pane, Pane 2.

As shown in the code sample below, you can also use the setPaneDimensions method to handle changing the pane sizes when the orientation of a device changes, for example, turning an iPad from landscape to portrait. Whenever the orientation of a device changes, sizeChanged() is called.

```
public void sizeChanged(int new_width, int new_height)
{
  // IMPORTANT to do this first to enable internal handling
  // that needs to happen when the canvas size changes.
  super.sizeChanged (new_width, new_height);

 setPaneDimensions ();
}
```

When to Use Views or Panes

As well as using panes to manage nUIViewDisplay objects within a nUIWindowDisplay, you can also nominate views as "side views". These can be used to provide a pop-up "side menu" or "toolbox"-style functionality for an application. The table below contrasts the use of panes and side views. As a general rule, side views are more suitable for phone screens than tablet devices, but are available for both.

Side View	Pane
Pops up when needed.	Is always open.
Obscures or displaces other content when open (may include nUINavViews).	Exists in its own space within the window.

Side View	Pane
Can be defined once to provide a global pop-up "toolbox" for the entire application.	Usually changes content throughout the application's life-cycle.
Only 2 side views per window possible at the same time (left and right).	Multiple panes possible.
Fixed x/y positions and height (width configurable).	Arbitrary layouts possible.
More space-efficient with smaller devices.	Better side-by-side layout of data and controls for larger devices.
Blocks interaction in other views when visible.	Allows for interaction across multiple views concurrently.

JavaScript Bridge

You can exchange messages between compiled Java code and a running JavaScript instance inside a nUIWebView or nUIWebviewElement.

Note: While most JavaScript engines implement a wide set of common functionalities, you must pay attention to differences between the various platforms. Mobile Designer does not adjust your HTML or JavaScript code to make it more compatible.

Maintaining Good Security

You must be aware that exchanging messages between JavaScript and Java can have various security implications. Therefore, you must consider which web pages may be loaded inside a nUIWebView or nUIWebviewElement, which messages may potentially be passed to and from that page, and how they are handled. You may consider implementing some or all of the items on this list:

- Checking Object.equals() on the nUIWebView or nUIWebviewElement making calls from JavaScript into Java.
- Using processURL() callbacks to create a URL whitelisting system and/or tracking the currently loaded web page.
- Passing a secret token into JavaScript from Java or an external server before accepting calls from JavaScript back into Java.
- Disabling callbacks with a boolean until they are expected.

- Obfuscating JavaScript code for release builds.

Sending a Message to JavaScript from Java

Messages are sent using the callJavaScript() method on the nUIWebView or nUIWebviewElement. You must specify the name of a JavaScript function to call and an array of java.lang.String objects for the functions's arguments. If no arguments are required, a string array of length 0 (zero) should be used. This is an example for two JavaScript functions, myFirstMethod() and mySecondMethodWithArgs(param1, param2):

```
//call myFirstMethod() to do something
myWebView.callJavaScript("myFirstMethod", new String[]{});
//call mySecondMethodWithArgs with "one" and "two" as arguments
myWebView.callJavaScript("mySecondMethodWithArgs", new String[]{"one", "two"});
```

Using callJavaScript() in this manner will cause execution on the Java side to wait for a return value. You can also pass an additional parameter to callJavaScript() that will allow the JavaScript function to return a string value to Java. Pass in a reference to a class that implements the NativeUI interface com.softwareag.mobile.runtime.nui.IJSCallback.

Evaluating an Arbitrary Chunk of JavaScript Code

With callJavaScript(), messages are passed from Java into an existing method hosted on the JavaScript side. Sometimes, this can be unsuitable for the application, and only a few simple lines of JavaScript need to be evaluated. For this, the method evaluateJavaScript() can be called.

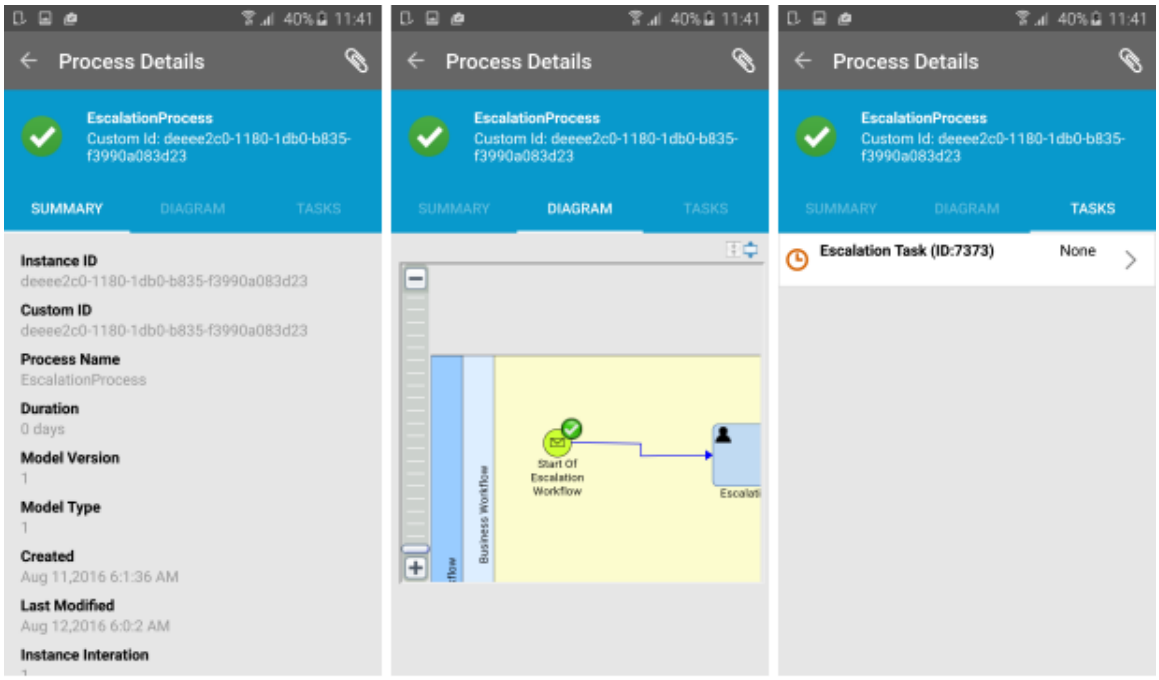
As with callJavaScript(), there are two ways to call evaluateJavaScript(). One is to take a string containing the JavaScript code, run synchronously, and return a string result (if applicable). The other method will take an additional object conforming to com.softwareag.mobile.runtime.nui.IJSCallback, and run asynchronously, returning any result at a later time.

Sending a Message to Java from JavaScript

Sometimes, events inside the browser may require the support of additional Java code, either for speed or to add functionality. For this purpose, Mobile Designer provides an additional callback method in nUIWebviewCallbackExt that you can implement. This method is called onJavaScriptCallback(). After creating a nUIWebviewCallbackExt and associating it with a nUIWebView or nUIWebviewElement, this callback can be accessed through the JavaScript function MDInterface.javaScriptCallback(clazz, method, parameters). It is expected that clazz and method will be directly convertable from a JavaScript-style var into a java.lang.String, and the parameters argument will become an array of strings. On the Java side, you must implement the logic required to handle the onJavaScriptCallback() method. Calls coming from the JavaScript side will be routed directly to this method. Although the parameters passed suggest the use of class names and methods, this is not mandatory, and you can implement logic that differs from this pattern.

Using Tabbed Views

You can create a tabbed view (also known as a segmented view) using the nUITabView object. This type of view allows an application to switch between different but related NativeUI views using a simple left/right swipe gesture. It can be used in place of a standard nUIViewDisplay as the child of a nUIWindowDisplay. For a visual reference, Google provide some images at "https://material.google.com/components/tabs.html" that illustrate the concept.



A nUITabView with 3 tabs

Each individual tab in a nUITabView contains two components. First, the content in form of a standard nUIViewDisplay object with the tables, buttons, etc. needed to lay out the application's content. Second, the label which describes the content of the tab. On Android and iOS, this may be a simple text such as "Network", an LCDUI image, or both. Complex arrangements can be created with nUITableButton.

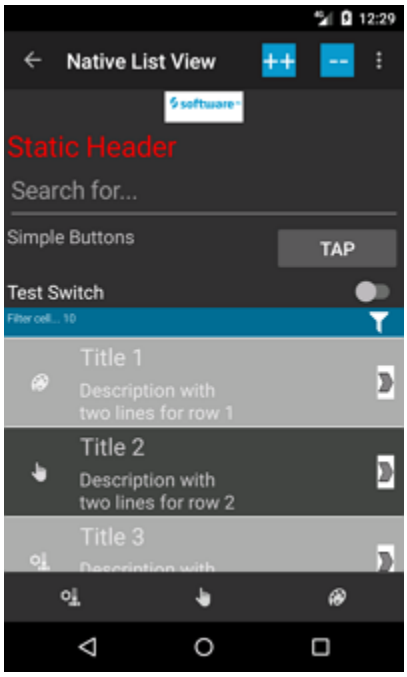
Integration in Mobile Designer

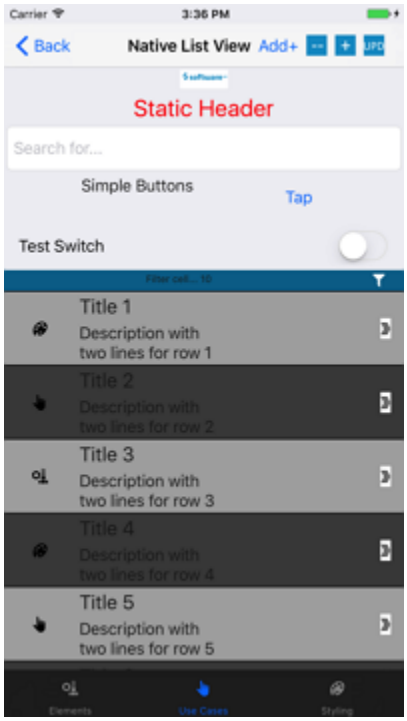
Content is provided to a nUITabView through an ITabViewProvider interface. You must create a class that implements this interface and assign it to an instance of nUITabView. The nUITabView object queries the methods in ITabViewProvider at run time to determine how to draw its contents. The getNumberOfTabs() method is called to determine how many tabs are required. This method is called only once, near the creation time of the native objects on-screen. For every tab, createTabElement(int index) is called to create the tab's label, and createTabView(int index) is called to create the tab's content. A nUITabViewListener can be

used to get notifications when a tab is selected. For example code for nUITabView, see the _NativeUIDemoNew_ Sample Project.

Using List Views and Elements

You can display long lists using two classes that conform to IListRenderer: nUIListView and nUIListElement. These list objects allow an application to load and display long lists in a more efficient manner.





To use nUIListElement or nUIListView, you must implement two interfaces and assign them to the list view:

- IListProvider to provide data to the list view.
- IListListener to get notifications from the list view. For detailed information, see *webMethods Mobile Designer Java API Reference*.

The PullToRefresh method is also available. By default, this functionality is disabled. Call the enablePullToRefresh(true) method to enable it. When you pull to refresh, the onRefresh() method is called. With this method, a bigger amount of data can be loaded in an asynchronous way and thereby prevent that the user interface is blocked. When the data is loaded, you can hide the top spinner by calling hideSpinner(IListListener.POSITION_TOP) and notify the list view that data was changed by calling the update() method. For detailed information, see the example in the com.softwareag.mobile.nativeuidemo.view.ListView class of the _NativeUIDemoNew_ project.

With the current API, you can easily implement endless scrolling. At first, configure the list view to notify you if the scrolling process has reached a specified amount of remaining elements and therefore more data can be loaded. For example, by calling setScrollThreshold(10), you are informed when less than 10 elements can be scrolled. When this point is reached, onScrollThresholdReached() is called. With this method, you can show the bottom spinner and load more data in an asynchronous way. When the data is loaded, you must hide the bottom spinner with hideSpinner(IListListener.POSITION_BOTTOM) and notify the list view that new data is available by calling the update() method.

For a better performance, use special methods for inserting, deleting, and updating rows instead of using the update() method.

Swiping Behavior

To be able to use swipe gestures within nUIListView and nUIListElement, you must define a swipe behavior for a particular cell type and swipe direction by implementing the getSwipeBehavior method in the IListProvider interface.

You can choose between

- SwipeToDeleteBehavior - Allows you to delete a cell by swiping.
- SwipeToEditBehavior - Allows you to show action buttons in a cell by swiping. However, the cell is not deleted.
- SwipeActionBehavior - Allows you to show action buttons in a cell and to delete this cell by swiping.

Use the factory class SwipeBehavior to create the required SwipeBehavior instance.

Note: You cannot assign different swipe behaviors to the same combination of cell type and swipe direction.

For detailed information, see the example in the com.softwareag.mobile.nativeuidemo.view.ListView class of the _NativeUIDemoNew_ project.

Using Edit Mode in List Views

If you want to select multiple rows in a list view, you can enter edit mode by calling the IListRenderer.startEditMode(IListEditModeListener listener) method. It returns an instance of the IListEditMode interface that contains some useful methods, such as

- selectRow(int row) - Selects a row.
- deselectRow(int row) - Deselects a row.
- finish() - Finishes edit mode.
- cancel() - Cancels edit mode.

In addition, you can determine if a specific row is selectable by implementing the IListProvider.shouldselectInEditMode(int row) method.

Using Element Identifiers

It can be tedious to test applications manually. A number of UI automation tools, such as Appium, have become available. Mostly, these tools enable you to run test scripts to access UI elements on the screen in a number of ways. Commands may be available to press or swipe at literal pixel positions, find elements according to their index in a parent object, or search for elements by a unique name assigned by the application developer and the item's type.

Mobile Designer provides a new Element Identifier API. This allows you to annotate your NativeUI objects with a unique name so that they can be detected by UI automation tools. For Android devices, the `content description` field of an element can be set, and on iOS, the `accessibility identifier` is used.

Note: It can be important to set an element's identifier before the element is drawn on the screen for the first time. For some elements, it is not always possible to update the value later.

For most elements, the setElementIdentifier() method is used directly:

```
nUIButtonElement myButton = new nUIButtonElement(-1, "Save Preferences");
myButton.setElementIdentifier("SavePrefs");
...
aView.add(myButton);
```

When annotating certain other elements, the methods to do so change slightly. Defining panes in windows is done as follows:

```
nUIWindowDisplay myWindow = new nUIWindowDisplay(-1);
myWindow.setPaneDimensions(2, sidePaneDimensions, "SideMenuPane");
```

Similarly, updating pop-up menus is done as follows:

```
nUIPopupMenuBuilder builder = new nUIPopupMenuBuilder();
builder.addItem("List Tasks", new TaskLister(), "StartListTasks");
builder.addItem("Update Tasks", new TaskUpdater(), "StartUpdateTasks");
```

For the following elements, the setElementIdentifier() method is not supported:

- nUISearchNavButton - Not supported on iOS.
- nUISeparatorElement - Not supported on Android.
- nUISpacerElement - Not supported on iOS.
- nUITabView - Not supported on iOS.
- nUITabElement - Not supported on Android.
- nUIWebviewElement and nUIWebView - Not supported on Android.

For further information, see *webMethods Mobile Designer Java API Reference*.

Note: Mobile Development sets element identifiers automatically using the name of the elements from the application model.

2 Native User Interface (NativeUI) Objects

- About the NativeUI Objects 51
- nUIAlertDialog 51
- nUIButtonElement 52
- nUICheckboxButton 53
- nUIContainerElement 54
- nUIDateEntry 55
- nUIDialogWindow 56
- nUIDisplayObject 57
- nUIDropdownlistEntry 58
- nUIElementDisplay 58
- nUIEntryElement 59
- nUIFloatingEntry 60
- nUIImageElement 61
- nUIListElement 61
- nUIListView 62
- nUINavbuttonElement 62
- nUINavView 65
- nUIObject 66
- nUIPopupMenuBuilder 66
- nUIProgressanimElement 67
- nUIRadioCheckbox 68
- nUISearchEntry 69
- nUISearchNavButton 70
- nUISeparatorElement 70
- nUISpacerElement 71
- nUISwitchButton 71

- nUITableButton 72
- nUITablecellElement 73
- nUITableElement 73
- nUITablerowElement 75
- nUITabView 75
- nUITextfieldElement 76
- nUITimerObject 77
- nUIViewDisplay 77
- nUIWebView 78
- nUIWebviewCallBack 80
- nUIWebviewElement 80
- nUIWindowDisplay 82

About the NativeUI Objects

The webMethods Mobile Designer native user interface (NativeUI) library provides a standard way to create user interfaces for mobile applications that run on multiple platforms. The NativeUI library is made up of NativeUI objects.

The NativeUI library includes platform-specific support for Android and iOS. When the NativeUI has platform-specific support, NativeUI maps each NativeUI object to a platform-specific object for a target device. As a result, when the user interface is rendered on a target device, the user interface displays using the platform-specific object. For example, you might want to include a check box in the mobile application's user interface. To do so, you can use the NativeUI object nUICheckboxButton. The nUICheckboxButton object maps to:

- android.widget.CheckBox for an Android device
- UISwitch for an iOS device

This Mobile Designer documentation describes the NativeUI objects in the NativeUI library. For additional information about the NativeUI objects, see information about the com.softwareag.mobile.runtime.nui package in the *webMethods Mobile Designer Java API Reference*.

Naming Conventions for NativeUI Objects

The names of the NativeUI objects begin with the prefix "nUI", followed by the object's name, which is then followed by the name of the object's parent. For example, the NativeUI check box object is a subtype of the NativeUI button object. Its name is nUICheckboxButton, where the object's name is Checkbox and parent name is Button.

Font Sizes Used Text in the NativeUI Objects

When displaying text in a NativeUI object, the font size of the text is based on the platform's user interface guidelines and usability requirements. However, some elements allow you to override the font size used on a per-object basis expressed as density independent points. Commonly, you use the setFontSize(float points) method.

nUIAlertDialog

com.softwareag.mobile.runtime.nui.nUIAlertDialog

Use to display a small pop-up that contains information. Use the pop-up to:

- Present information to the user.
- Interact with the user by displaying a simple question, for example, a question requiring a "yes" or "no" answer.

Usage Notes

- Include at least one button in a nUIAlertDialog object.
- The following are platform-specific considerations:

Android	Android devices support no more than three buttons and ignore additional buttons.
iOS	When using more than two buttons, iOS devices stack the buttons vertically in an alert dialog. Software AG recommends limiting the number of buttons to four or five.

Example

This code sample displays an alert dialog with two buttons. Details on how the example code is rendered on various platforms follow the code sample.

```
nUIAlertDialog alertDialog = new nUIAlertDialog
(
NUIID_MY_ALERT_DIALOG,
"Lorem Ipsum",
"Dolor sit amet?",
new String[]{"Lorem", "Ipsum"},
new int[]{NUIID_BUTTON_LOREM, NUIID_BUTTON_IPSUM}
);
```

Platform	Platform-Specific Class and Visual Reference
Android	Dialogs and the android.app.AlertDialog "https://developer.android.com/guide/topics/ui/dialogs"
iOS	UIAlertView "https://developer.apple.com/design/human-interface-guidelines/ios/views/alerts/"

nUIButtonElement

com.softwareag.mobile.runtime.nui.nUIButtonElement

Use to display a single button that contains a text label.

Usage Notes

- Based on the platform, the nUIButtonElement object exhibits different behavior and appearance.

- The following lists the default horizontal text alignment of the button label based on the platform:

Android	Button label is aligned left
iOS	Button label is aligned left

Example

This code sample displays a button. Details on how the example code is rendered on various platforms follow the code sample.

```
view.add(new nUIButtonElement(NUIID_MY_BUTTON, "nUIButtonElement"));
```

Platform	Platform-Specific Class and Visual Reference
Android	android.widget.Button "https://developer.android.com/guide/topics/ui/controls/button"
iOS	UIButton "https://developer.apple.com/design/human-interface-guidelines/ios/controls/buttons/"

nUICheckboxButton

com.softwareag.mobile.runtime.nui.nUICheckboxButton

Use to display a check box.

Usage Notes

- Valid states for the check box are `0` (zero) meaning clear and `1` (one) meaning selected.
- The nUICheckboxButton object provides the following check box types. Use the type that is most appropriate for your application's target platforms.
  - `nUICheckboxButton.TYPE_DEFAULT`, which indicates the application uses the check box type that is considered the most appropriate for the target platform.
  - `nUICheckboxButton.TYPE_OFF_ON` indicates a check box that uses "On" and "Off".
  - `nUICheckboxButton.TYPE_YES_NO` specifies a check box that uses "Yes" and "No". For platforms that do not support yes/no check boxes, Mobile Designer implements a nUIButtonElement object with equivalent "Yes" and "No" text labels.

The default state for a check box is `0` (zero), meaning clear, off, or no.

Example

This code sample displays a check box. Details on how the example code is rendered on various platforms follow the code sample.

```
view.add(new nUICheckboxButton(NUIID_MY_CHECKBOX, "nUICheckboxButton"));
```

Platform	Platform-Specific Class and Visual Reference
Android	android.widget.CheckBox "https://developer.android.com/guide/topics/ui/controls/checkbox"
iOS	UISwitch "https://developer.apple.com/design/human-interface-guidelines/ios/controls/switches/"

nUIContainerElement

com.softwareag.mobile.runtime.nui.nUIContainerElement

Use to display a container that holds other NativeUI objects.

You can set the container's attributes to allow scrolling. For example, the application might use the container to hold long pieces of text that exceeds the viewable area, allowing the user to scroll through the text.

Usage Notes

- Set the `Height` attribute to set height of the object. By default, the nUIContainerElement object occupies the remaining width of the parent object. However, you can adjust the width of the nUIContainerElement object using the `Width` attribute.
- If you want to use a scrolling nUIContainerElement object in a view, ensure that the parent nUIViewDisplay object does not allow scrolling.

  Caution: Setting the `Hscrollable` attribute to `true` to allow horizontal scrolling currently results in undefined behavior.

- The following are platform-specific considerations:

  Android The `InnerX` and `InnerY` default values are 0 (zero).

  iOS The `InnerX` and `InnerY` default values are 0 (zero).

Example

This code sample displays the nUIContainerElement object between two nUIButtonElement objects. Details on how the example code is rendered on various platforms follow the code sample.

```
//Add buttons to help demonstrate the bounds of the container.
view.add(new nUIButtonElement(-1, "nUIButtonElement 1"));

nUIContainerElement my_container = new nUIContainerElement(-1);
my_container.setHeight(150);
my_container.add(new nUITextfieldElement(-1, LOREM_IPSUM_STRING));
view.add(my_container);

view.add(new nUIButtonElement(-1, "nUIButtonElement 2"));
```

Platform	Platform-Specific Class and Visual Reference
Android	android.widget.ScrollView with RelativeLayout.LayoutParams
iOS	UIView with a UIScrollView "https://developer.apple.com/design/human-interface-guidelines/ios/views/scroll-views/"

nUIDateEntry

com.softwareag.mobile.runtime.nui.nUIDateEntry

Use to display a date or time selector control.

Usage Notes

- Use the `Format` attribute to indicate whether you want a date or time selector control:

For this type of selector	Specify the following for the `Format` attribute
Date with day, month, and year	`nUIDateEntry.dd_MM_yyyy`
Time with hours and minutes	`nUIDateEntry.HH_mm`

- When getting the `Date` attribute after an `EVT_POST_EDIT` call to a nUIDateEntry object, only the information for the requested `Format` is valid. Data outside the specific `Format` is undefined.

Example

This code sample displays a date selector control. Details on how the example code is rendered on various platforms follow the code sample.

```
view.add(new nUIDateEntry(NUIID_MY_DATE, null)); //null = current date
```

Platform	Platform-Specific Class and Visual Reference
Android	DatePickerDialog "https://developer.android.com/guide/topics/ui/controls/pickers"
iOS	UIDatePicker "https://developer.apple.com/design/human-interface-guidelines/ios/controls/pickers/"

nUIDialogWindow

com.softwareag.mobile.runtime.nui.nUIDialogWindow

Use to display a pop-up window.

Usage Notes

- You can add a nUIViewDisplay object to the nUIDialogWindow object.
- The nUIDialogWindow object does not support multiple panes, the nUINavView object, or nUINavbuttonElement object.

Example

This code sample displays a pop-up window. Details on how the example code is rendered on various platforms follow the code sample.

```
nUIDialogWindow custom_dialog = new nUIDialogWindow(NUIID_MY_CUSTOM_DIALOG);
nUIViewDisplay view = new nUIViewDisplay(-1);
view.add(new nUITextfieldElement(-1, "Lorem"));
view.add(new nUIEntryElement(NUIID_ENTRYELEMENT_IPSUM, "Ipsum"));
view.add(new nUIEntryElement(NUIID_ENTRYELEMENT_DOLOR, "Dolor"));
nUITableElement button_table = new nUITableElement(-1, new int[]{50, 50});
nUITablerowElement tr = new nUITablerowElement(-1);
{
 nUITablecellElement tc = new nUITablecellElement(-1);
 {
  tc.add(new nUIButtonElement(NUIID_BUTTON_SIT, "Sit"));
 }
 tr.add(tc);
 tc = new nUITablecellElement(-1);
 {
   tc.add(new nUIButtonElement(NUIID_BUTTON_AMET, "Amet"));
 }
 tr.add(tc);
```

```
}
button_table.add(tr);
view.add(button_table);
custom_dialog.add(view);
```

Platform	Platform-Specific Class and Visual Reference
Android	android.app.Dialog "https://developer.android.com/guide/topics/ui/dialogs"
iOS	UIView initialized with a UIModalPresentationFormSheet **Note:** On an iPhone, the nUIDialogWindow object occupies the entire screen.

nUIDisplayObject

com.softwareag.mobile.runtime.nui.nUIDisplayObject

The nUIDisplayObject class is a base class for Display NativeUI objects. The nUIElementDisplay, nUIViewDisplay, and nUIWindowDisplay classes extend the nUIDisplayObject class.

Usage Notes

- When specifying the `Width` and `Right` attributes, you can specify just one of these attributes. Mobile Designer infers the value of the attribute you do not define from the value of the defined attribute.
- When specifying the `Height` and `Bottom` attributes, you can specify just one of these attributes. Mobile Designer infers the value of the attribute you do not define from the value of the defined attribute.
- The `X` and `Left` coordinate position attributes are equivalent. You can specify just one of them.
- The `Y` and `Right` coordinate position attributes are equivalent. You can specify just one of them.
- The default value for inner padding attributes is `0` (zero).

  Display objects, such as tables and views, might override the default inner padding values.
- By default, the inner width and height values match the values for the `X` and `Y` padding on the `Right` and `Bottom` of the object.

  You can configure the object's `Width` and `Height` attribute values to prevent this duplication.

nUIDropdownlistEntry

com.softwareag.mobile.runtime.nui.nUIDropdownlistEntry

Use to display a drop-down list that contains selection items.

Usage Notes

- If a drop-down list contains less than five items, consider using the nUIRadioCheckbox object instead.

Example

This code sample displays a drop-down list. Details on how the example code is rendered on various platforms follow the code sample.

```
String[] list_items = new String[] {"Lorem", "ipsum", "dolor", "sit", "amet"};
view.add(new nUIDropdownlistEntry(NUIID_MY_DROPDOWNLIST, list_items));
```

Platform	Platform-Specific Class and Visual Reference
Android	android.widget.Spinner "https://developer.android.com/guide/topics/ui/controls/spinner"
iOS	UIPickerView "https://developer.apple.com/design/human-interface-guidelines/ios/controls/pickers/"

nUIElementDisplay

com.softwareag.mobile.runtime.nui.nUIElementDisplay

The nUIElementDisplay class is the base class for Element NativeUI objects. The nUIElementDisplay class overrides the parent's `Y` attribute and inner padding values.

By default, Element NativeUI objects display one below the other in their parent unless the application specifically position the Element objects.

nUIEntryElement

com.softwareag.mobile.runtime.nui.nUIEntryElement

Use to display a text entry box. You can restrict the user input to alphanumeric characters or only numbers. You can mask the field's contents, making the field suitable for a user to enter passwords or personal identifier numbers (PIN)s.

Usage Notes

- Set the `Format` attribute to indicate the type of text allowable in the text entry field. Use one of the following values:

Value	Meaning
`nUIEntryElement.FORMAT_STRING_ANY`	Alphanumeric field
`nUIEntryElement.FORMAT_NUMBER_ANY`	Numeric-only field
`nUIEntryElement.FORMAT_STRING_HIDDEN`	Hidden (masked) alphanumeric password field
`nUIEntryElement.FORMAT_PIN_HIDDEN`	Hidden (masked) PIN field

- Use the `HintText` attribute to provide text to indicate what the user should enter in the entry field. The text displays in the entry field, typically in a light gray, and disappears as soon as the user starts typing in the field.

  The following lists platform considerations:

  iOS Hint text does not display in multi-line entry boxes.

Example

This code sample displays a text box. Details on how the example code is rendered on various platforms follow the code sample.

```
view.add(new nUIEntryElement(NUIID_MY_ENTRY, "nUIEntryElement"));
```

Platform	Platform-Specific Class and Visual Reference
Android	android.widget.EditText
iOS	UITextField for a single-line text box

Platform	Platform-Specific Class and Visual Reference
	"https://developer.apple.com/design/human-interface-guidelines/ios/controls/text-fields/"
	UITextView for a multi-line text box
	"https://developer.apple.com/design/human-interface-guidelines/ios/views/text-views"

nUIFloatingEntry

com.softwareag.mobile.runtime.nui.nUIFloatingEntry

Use to provide a text-entry element that allows any hint-text set to float above the user-entered text as a label.

Usage Notes

- For Android devices, the `error color` and `label color` attributes are ignored, and the system-defaults are used. You can modify these on an app-wide basis with XML styles. The space allocated to prefix and postfix hint-texts or images is identical on both sides and therefore is influenced by the larger of the two values.
- With iOS devices, the `error text` attribute occupies the same space as the floating label. This means that both cannot be visible at the same time. When enabled, the error text will always take priority over the floating label.

Example

This code sample creates an empty nUIFloatingEntry and sets the floating hint to "Family Name". An IStringValidator is used to detect `@` characters inside the string and switch the nUIFloatingEntry into error mode if any are detected. Details on how the example code is rendered on various platforms follow the code sample.

```
nUIFloatingEntry fe = new nUIFloatingEntry("");
fe.setHintText("Family Name");
fe.setErrorText("No '@' allowed here.");
fe.setStringValidator(new IStringValidator()
{
 public boolean accept(String s)
 {
  return (s.indexOf('@') < 0);
 }
});
view.add(fe);
```

Platform	Platform-Specific Class and Visual Reference
Android	TextInputLayout with a TextInputEditText

Platform	Platform-Specific Class and Visual Reference
	"https://material.io/develop/android/components/text-input-layout/"
iOS	Custom implementation based around JVFloatLabeledTextField and JVFloatLabelledTextView "https://github.com/jverdi/JVFloatLabeledTextField"

nUIImageElement

com.softwareag.mobile.runtime.nui.nUIImageElement

Use to display an image.

Usage Notes

- You can make a simple image behave like a button and generate `EVT_TRIGGER` events by calling setTriggerable(true) when creating the image.
- The following are platform-specific considerations:

Android	The nUIImageElement object is the android.widget.ImageView class.
iOS	The nUIImageElement object is the UIImage class. For more information, see iOS Developer Library's "Drawing and Creating Images".

nUIListElement

com.softwareag.mobile.runtime.nui.nUIListElement

Use to add a scrollable list of elements to the view. Elements are fetched on an as-needed basis.

Example

This code sample instantiates the user's IListViewProvider class, MyListProvider, and then uses it for the creation of a nUIListElement with a fixed height. Details on how the example code is rendered on various platforms follow the code sample.

```
MyListProvider provider = new MyListProvider();
          nUIListElement list = new nUIListElement(provider);
          list.setHeight(400);
          view.add(list);
```

Platform	Platform-Specific Class and Visual Reference
Android	RecyclerView
iOS	Custom implementation based around UITableView

nUIListView

com.softwareag.mobile.runtime.nui.nUIListView

Use to add a view that contains a single nUIListElement , filling the entire space available to it.

Example

This code sample instantiates the user's IListViewProvider class, MyListProvider, and then uses it for the creation of a nUIListView with header text and a back button. Details on how the example code is rendered on various platforms follow the code sample.

```
MyListProvider l_provider = new MyListProvider();
          nUIListView list_view = new nUIListView(-1, l_provider);
          list_view.setHeadertext("A nUIListView");
          nUINavbuttonElement ne = new nUINavbuttonElement(
            NUIID_BACK_BUTTON, "Back",
          nUINavbuttonElement.TYPE_BACK, null);
          list_view.add(ne);
```

Platform	Platform-Specific Class and Visual Reference
Android	RecyclerView
iOS	Custom implementation based around UITableView

nUINavbuttonElement

com.softwareag.mobile.runtime.nui.nUINavbuttonElement

Use to display a button within a nUIViewDisplay object or a nUINavView object.

Usage Notes

- To use the nUINavbuttonElement object as a Back button, set the nUINavbuttonElement object's `Type` attribute to `nUINavbuttonElement.TYPE_BACK`.

  Note: Do *not* use the nUINavbuttonElement object as a Back button in a nUINavView.

- When using the nUINavbuttonElement object as a Back button in a nUIViewDisplay object, you do not need to define an icon to use for the nUINavbuttonElement object.

  When a user presses a physical Back button or touches a Back area on a touch screen, the nUINavbuttonElement object reacts to the user interaction.
- When using the nUINavbuttonElement object as a button in a nUINavView object, define the `Text` and `Icon` attributes for the button.
- When specifying an image with the `Icon` attribute, the file must be a PNG file. For information about using graphics in a mobile application, see "Preparing Graphics for Your Mobile Application" and "Icon Creation and Usage for webMethods Mobile Designer".
- The following are platform-specific considerations:

Android

- The nUINavbuttonElement object is a android.view.MenuItem class. For more information, see "http://developer.android.com/guide/topics/ui/menus.html".
- When you use the nUINavbuttonElement object in a nUINavView object, at least the `Text` or the `Icon` attributes must be supplied.
- When you use the nUINavbuttonElement object in a nUIViewDisplay object:
  - The button displays in the header bar. This support is only available if `android.nativeui.view.header.version` is set to the newer style header bar introduced with Ice Cream Sandwich.
  - If you set the `Type` attribute to `nUINavbuttonElement.TYPE_BACK`, and if the Android device has a physical Back button, the Back button is used. Otherwise, if the newer style header bar introduced with Ice Cream Sandwich is in use, Android draws a Back button in the header.
  - If you use the `HeaderText` attribute, and there is no space available for both the header text and the nUINavbuttonElement objects, the header text is truncated.
  - It is recommended that you insert no more than three nUINavbuttonElement objects in a nUIViewDisplay object.
  - The `Icon` attribute is required. The icon will be used if it is supplied. If not, the `Text` attribute is used instead. Provide a brief text for the attribute as the text will be

cropped at a width 1.5 times the width of the standard icon width.

- The size of the image you specify with the `Icon` attribute for a pop-up menu is based on the density of the screen of the device:
  - Low density (ldpi) images is 36x36 pixels.
  - Medium density (mdpi) images is 48x48 pixels.
  - High density (hdpi) images is 72x72 pixels.

  Note: Extra high density (xdpi) is not applicable.
- The size of the image you specify with the `Icon` attribute for an action bar is based on the density of the screen of the device:
  - Medium density (mdpi) images is 32x32 pixels.
  - High density (hdpi) images is 48x48 pixels.
  - Extra high density (xdpi) images is 64x64 pixels.

  Note: Low density (ldpi) is not applicable.

iOS

- The nUINavbuttonElement object is a UITabBarItem class.
- When you use the nUINavbuttonElement object in a nUINavView object:
  - The `Text` and `Icon` attributes are required.
  - When specifying an image with the `Icon` attribute, the image should be 32x32 pixels for a non-retina display or 64x64 pixels for a retina display. Additionally, the image should be black or transparent. The iOS device automatically adds highlighting and gradients at run time.
- When you use the nUINavbuttonElement object in a nUIViewDisplay object:
  - The button displays in the header bar.
  - If you set the `Type` attribute to `nUINavbuttonElement.TYPE_BACK`, the button displays in the left part of the header.
  - The `Icon` or `Text` attribute is required. If you specify both, `Icon` is used.

- If you specify only the `Text` attribute, the iOS devices draws the text inside a button.
- When specifying an image with the `Icon` attribute, it is recommended that the image be 24x24 pixels for a non-retina display or 48x48 pixels for a retina display. The iOS device draws the icon without additional borders.
- If you use the `HeaderText` attribute and space is not available for both the header text and the nUINavbuttonElement objects, the header text is truncated.
- For an iPhone running on iOS, it is recommended that you insert no more than three nUINavbuttonElement objects in a nUIViewDisplay object.
- For an iPad running on iOS, it is recommended that you insert no more than six nUINavbuttonElement objects in a nUIViewDisplay object.

nUINavView

com.softwareag.mobile.runtime.nui.nUINavView

Use to display the navigation view.

The navigation view has different formats based on the platform. For example, for some platforms the navigation view might display as a menu bar that is always visible and uses both icons and text. For other platforms, the navigation view might have hidden menu items that are displayed only when a user presses a button.

Usage Notes

- You can add nUINavbuttonElement objects to the nUINavView object.

  Note: The nUINavbuttonElement object should *not* represent Back buttons. That is, the object's `Type` attribute should *not* be set to `nUINavbuttonElement.TYPE_BACK`.

- The following are platform-specific considerations:

Android

- It is recommended to limit the number of buttons you add to a nUINavView object to 6 buttons. If you add more than 6 buttons, a More button displays on some Android devices.

iOS

- It is recommended to limit the number of buttons you add to a nUINavView object on an iPhone in full-screen portrait mode to 8 buttons.

Example

This code sample displays a navigation view. Details on how the example code is rendered on various platforms follow the code sample.

```
protected nUIViewDisplay onCreateMainNavbarView()
{
//loadImage calls Image.createImage() with the appropriate path and file
extension.
  navbar_view = new nUINavView(NUIID_NAV_VIEW);
  navbar_view.setVscrollable(false);
  navbar_view.setBgcolor (0);
  navbar_view.add(new nUINavbuttonElement(-1, "Lorem",
                                          loadImage ("ChartArea")));
  navbar_view.add(new nUINavbuttonElement(-1, "Ipsum",
                                          loadImage ("ChartPie")));
  navbar_view.add(new nUINavbuttonElement(-1, "Dolor",
                                          loadImage ("ChartBar")));
  return navbar_view;
}
```

Platform	Platform-Specific Class and Visual Reference
Android	Custom code to emulate a split action bar "http://developer.android.com/design/patterns/actionbar.html"
iOS	UITabBar "https://developer.apple.com/design/human-interface-guidelines/ios/bars/tab-bars/"

nUIObject

com.softwareag.mobile.runtime.nui.nUIObject

The nUIObject class is the base class for all NativeUI objects. Do not instantiate this object directly.

nUIPopupMenuBuilder

com.softwareag.mobile.runtime.nui.nUIPopupMenuBuilder

Use to display a lightweight multiple-choice pop-up menu.

Usage Notes

- Unlike other NativeUI widgets, the pop-up menu is created indirectly through a builder. This builder does not need to be placed anywhere in the user interface hierarchy.
- Based on the platform and the available screen size, the pop-up either appears as a menu attached to a nominated on-screen widget, or at the bottom of the screen. Passing the anchor element is mandatory. Calls to show() with null have no effect.
- Each element inside the menu has a text label and an associated action. If the action is null, the menu is closed without taking any actions.
- A Cancel button is available on some platforms. The text of this button can be changed.

Example

This code sample displays a simple pop-up menu with three items. Details on how the example code is rendered on various platforms follow the code sample.

```
nUIPopupMenuBuilder builder = new nUIPopupMenuBuilder();
//Adding some items to the popup
builder.addItem("Do Something", new DoSomethingRunnable());
builder.addItem("Do Another Thing", new DoAnotherThingRunnable());
builder.addItem("Do Nothing", null);
...
// Later on, we can show the popup.
// We're anchoring to the button that triggered it...
//nUIButtonElement "aNuiButton" is defined elsewhere.
builder.show(aNuiButton);
```

Platform	Platform-Specific Class and Visual Reference
Android	android.widget.PopupMenu "https://developer.android.com/reference/android/widget/PopupMenu"
iOS	UIAlertAction or UIActionSheet, depending on OS version "https://developer.apple.com/design/human-interface-guidelines/ios/views/action-sheets/"

nUIProgressanimElement

com.softwareag.mobile.runtime.nui.nUIProgressanimElement

Use to display an animated status indicator that an application can display to indicate background activity is in progress.

Example

This code sample displays a status indicator. Details on how the example code is rendered on various platforms follow the code sample.

```
view.add(new nUIProgressanimElement(NUIID_MY_PROGRESSANIM));
```

Platform	Platform-Specific Class and Visual Reference
Android	android.widget.ProgressBar
iOS	UIActivityIndicatorView "https://developer.apple.com/design/human-interface-guidelines/ios/controls/progress-indicators/"

nUIRadioCheckbox

com.softwareag.mobile.runtime.nui.nUIRadioCheckbox

Use to display a single radio button that uses two states, selected or cleared.

Usage Notes

- You can place a radio button in a group with other radio buttons and enable the user to select only one of the available radio button options. To do so, set the `GroupID` attribute of the buttons in the group to the same value.

Example

This code sample displays a radio button. Details on how the example code is rendered on various platforms follow the code sample.

```
view.add(new nUIRadioCheckbox(NUIID_MY_RADIOCHECKBOX, "nUIRadioCheckbox"));
```

Platform	Platform-Specific Class and Visual Reference
Android	android.widget.RadioButton "https://developer.android.com/guide/topics/ui/controls/radiobutton"
iOS	UIButton

nUISearchEntry

com.softwareag.mobile.runtime.nui.nUISearchEntry

Use to display a search entry field. You can restrict the user input to alphanumeric characters or only numbers.

Note: The nUISearchEntry NativeUI object is similar to the nUIEntryElement NativeUI object except that the nUISearchEntry object does not allow masking the entry field.

Usage Notes

- Set the `Format` attribute to indicate the type of text allowable in the text entry field. Use one of the following values:

Value	Meaning
`nUIEntryElement.FORMAT_STRING_ANY`	Alphanumeric field
`nUIEntryElement.FORMAT_NUMBER_ANY`	Numeric-only field

- The entered text is always visible in the search entry field. If you need to mask the field's contents, for example to use for a field where a user enters passwords or personal identifier numbers (PIN)s, use nUIEntryElement object instead.
- Use the `HintText` attribute to provide text to indicate what the user should enter in the entry field. The text displays in the entry field, typically in a light gray, and disappears as soon as the user starts typing in the field.
- The following are platform-specific considerations:

Android

- Android devices generate an `EVT_TRIGGER` event when a user selects the button adjacent to the text field.

iOS

- iOS devices generate an `EVT_TRIGGER` event when a user presses the Search or Enter key.
- Hint text does not display in multi-line entry boxes.

Example

This code sample displays a search entry field. Details on how the example code is rendered on various platforms follow the code sample.

```
view.add(new nUISearchEntry(NUIID_MY_SEARCH_ENTRY, "nUISearchEntry"));
```

Platform	Platform-Specific Class and Visual Reference
Android	android.widget.SearchView "https://developer.android.com/guide/topics/search/"
iOS	UISearchBar "https://developer.apple.com/design/human-interface-guidelines/ios/bars/search-bars/"

nUISearchNavButton

com.softwareag.mobile.runtime.nui.nUISearchNavbutton

Use to display an expandable search field within a nUIViewDisplay object. By default, only a search icon is visible in the header which can be tapped to show/expand the search field.

Use ISearchFieldListener to get notified when text was changed, or the Submit or Cancel buttons were clicked.

For more information, see *webMethods Mobile Designer Java API Reference*.

Example

```
ISearchFieldListener listener = <your_implementation>;
nUIViewDisplay view = <current view>;
nUISearchNavButton searchField = new nUISearchNavButton();
searchField.setSearchFieldListener(listener);
searchField.setSearchFieldHintText("Enter search query…");
view.add(searchField);
```

nUISeparatorElement

com.softwareag.mobile.runtime.nui.nUISeparatorElement

Use to display a horizontal line that separates blocks of content.

Usage Notes

- Setting the `Height` attribute alters the padding above and below the horizontal, separator line. It does not alter the size of the line itself.

Example

This code sample displays a horizontal, separator line. Details on how the example code is rendered on various platforms follow the code sample.

```
view.add(new nUISeparatorElement(-1))
```

Platform	Platform-Specific Class and Visual Reference
Android	Custom android.view.View
iOS	UIView customized with a horizontal rule between content blocks

nUISpacerElement

com.softwareag.mobile.runtime.nui.nUISpacerElement

Use to add blank space between NativeUI objects to create extra padding.

Usage Notes

- Use the `Height` attribute to set the appropriate size for the blank space.
- You can set the height directly using setHeight().

nUISwitchButton

com.softwareag.mobile.runtime.nui.nUISwitchButton

Use to display a single switch that may optionally have a text label.

Usage Notes

- Based on the platform, the nUISwitchButton exhibits different behavior and appearance.
- The nUISwitchButton will always have two states, `on` (true) and `off` (false).
- The default state of a nUISwitchButton is `off`, which corresponds to 0 (zero), false, or clear.
- Additional methods are provided to change the state of the nUISwitchButton that accept and return boolean values.

Example

This code sample displays two types of switch. The first has a text label, and the second does not. Details on how the labelled switches are rendered on various platforms follow the code sample.

```
view.add(new nUISwitchButton(NUIID_LABEL_SWITCH,"nUISwitchButton"));//with label
view.add(new nUISwitchButton(NUIID_SIMPLE_SWITCH)); //without label
```

Platform	Platform-Specific Class and Visual Reference
Android	android.support.v7.widget.SwitchCompat "https://developer.android.com/guide/topics/ui/controls/togglebutton"
iOS	UISwitch "https://developer.apple.com/design/human-interface-guidelines/ios/controls/switches/"

nUITableButton

com.softwareag.mobile.runtime.nui.nUITableButton

Use to display a table that contains other NativeUI objects and acts as a button.

Example

This code sample displays a nUITableButton object that contains an image and text.

```
nUITableElement table;
nUITablerowElement tr;
nUITablecellElement tc;

//loadImage calls Image.createImage() with the appropriate path
//and file extension.
Image person = loadImage("PersonRealisticSingle");
table = new nUITableElement(-1, new int[]{30,70});
{
  tr = new nUITablerowElement(-1);
  {
     tc = new nUITablecellElement(-1);
     {
        tc.add(new nUIImageElement(-1, person));
        tc.setVspan(2);
     tc.setVAlign(nUIConstants.center);
     }
     tr.add(tc);

     tc = new nUITablecellElement(-1);
     {
        nUITextfieldElement header = new nUITextfieldElement(-1, "Lorem
Ipsum.");
        header.setHAlign(nUIConstants.left);
        tc.add(header);
     }
     tr.add(tc);
  }
  table.add(tr);

  tr = new nUITablerowElement(-1);
  {
        //empty cell, cell above us spans into here.
```

```
        tc = new nUITablecellElement(-1);
        tr.add(tc);

        tc = new nUITablecellElement(-1);
        {
             nUITextfieldElement details = new nUITextfieldElement(-1, "Dolor
sit amet, consectetur adipisicing elit, sed do eiusmod tempor.");
             details.setMaxLines(2);
             details.setFontSize(nUIConstants.size_small);
             details.setHAlign(nUIConstants.left);
             tc.add(details);
        }
        tr.add(tc);
     }
     table.add(tr);
}
nUITableButton person_with_text =
          new nUITableButton(NUIID_MY_PERSON_TABLEBUTTON);
person_with_text.add(table);
view.add(person_with_text);
```

nUITablecellElement

com.softwareag.mobile.runtime.nui.nUITablecellElement

Use to add a table cell to a table row (nUITablerowElement).

Usage Notes

- The nUITablecellElement object sets the `InnerX` attribute to *screen_width* divided by 128, and `InnerY` attribute to *screen_height* divided by 256 on all platforms.
- A table cell can contain more than one child object. The table cell objects are positioned and aligned in a manner similar to a nUIViewDisplay object.
- Use the `Bgcolor` attribute to set the table cell's background color. To specify a color, set the `Bgcolor` attribute to an integer that is based on a 32-bit Alpha Red Green Blue (ARGB) format. You can use any solid color. If you specify an Alpha value that is not opaque (0xFF), the behavior of the table cell object is undefined. Setting a `Bgcolor` will override any previous `BackgroundDrawable` that has been set.
- Use the `BackgroundDrawable` attribute to set the table cell's background drawable. A background drawable may be a solid color, defined by a ColorBackground object, or an image defined by a PatternImage object. Specifying a background color with an alpha value that is not opaque `(0xFF)` will result in undefined behavior. Setting a `BackgroundDrawable` will override any previous `Bgcolor` that has been set.

nUITableElement

com.softwareag.mobile.runtime.nui.nUITableElement

Use to display a table that is composed of nUITablerowElement and nUITablecellElement objects.

Usage Notes

- Use the `Bgcolor` attribute to set the table's background color. By default, the background color is transparent. To change the color, set the `Bgcolor` attribute to an integer that is based on a 32-bit Alpha Red Green Blue (ARGB) format. You can use any solid color. If you specify an Alpha value that is not opaque (0xFF), the behavior of the table object is undefined. Setting a `Bgcolor` will overrride any previous `BackgroundDrawable` set.
- Use the `BackgroundDrawable` attribute to set the table cell's background drawable. A background drawable may be a solid color, defined by a ColorBackground object, or an image defined by a PatternImage object. Solid colors with an alpha value that is not fully opaque `(0xFF)` will exhibit undefined behavior. Setting a `BackgroundDrawable` will override any previous `Bgcolor` that has been set.
- Use the `RelWidths` attribute to specify an array of integer values that provide a relative width for each column.
- A table occupies the space available from the parent container, minus any padding. The actual pixel width of each column is determined using the following formula:

```
column_px_width =
  (table_px_width * column_rel_width) / sum_of_all_rel_widths
```

- Consider the width limitations of the target devices when determining the number of columns to use in a table.
- The `RelWidths` attribute can be updated after the table has been created, however, the number of columns in the table must remain constant.

Examples

- This code sample creates a table with two equal-width columns.

```
new nUITableElement(-1, new int [] { 1, 1 });
```

- This code sample uses percentage values to create a table with three columns, where the middle column is twice as wide as the first and last columns. The column percentages should add up to 100%.

```
new nUITableElement(-1, new int [] { 25, 50, 25 });
```

- This code sample creates a table using nUITableElement, nUITableButton, nUITablerowElement, and nUITablecellElement.

```
nUITableElement table;
nUITablerowElement tr;
nUITablecellElement tc;
table = new nUITableElement(-1, new int [] { 70, 30 });
{
   tr = new nUITablerowElement (-1);
   {
       tc = new nUITablecellElement (-1);
       {
           tc.add(new nUITextfieldElement (-1, "Lorem"));
       }
       tr.add (tc);
```

```
        tc = new nUITablecellElement (-1);
        {
            tc.add(new nUITextfieldElement (-1, "ipsum"));
        }
        tr.add (tc);
}
table.add (tr);

tr = new nUITablerowElement (-1);
{
        tc = new nUITablecellElement (-1);
        {
            tc.add(new nUIButtonElement (-1, "dolor"));
        }
        tr.add (tc);
        tc = new nUITablecellElement (-1);
        {
            tc.add(new nUIButtonElement (-1, "sit"));
        }
        tr.add (tc);
   }
   table.add (tr);
}
view.add(table);
```

nUITablerowElement

com.softwareag.mobile.runtime.nui.nUITablerowElement

Use to add a row to a table (nUITableElement). The table row contains one or more nUITablecellElement objects.

Usage Notes

- Use the `Bgcolor` attribute to set the table row's background color. By default, the background color is transparent. To change the color, set the `Bgcolor` attribute to an integer that is based on a 32-bit Alpha Red Green Blue (ARGB) format. You can use any solid color. If you specify an Alpha value that is not opaque (0xFF), the behavior of the table row object is undefined. Setting a `Bgcolor` will override any previous `BackgroundDrawable` set.
- Use the `BackgroundDrawable` attribute to set the table row's background drawable. A background drawable may be a solid color, defined by a ColorBackground object, or an image defined by a PatternImage. An alpha value that is not fully opaque `(0xFF)` will exhibit undefined behavior. Setting a `BackgroundDrawable` will override any previous `Bgcolor` that has been set.

nUITabView

com.softwareag.mobile.runtime.nui.nUITabView

Use to group multiple, similar views together.

Example

See _NativeUIDemoNew_ sample project for code. The following illustrates how the code is rendered on various platforms.

Platform	Platform-Specific Class
Android	android.support.design.widget.TabLayout and android.support.v4.view.ViewPager
iOS	Custom implementation based around the UIScrollView class

nUITextfieldElement

com.softwareag.mobile.runtime.nui.nUITextfieldElement

Use to display plain text in a label or for a block of text.

Usage Notes

- The following are platform-specific considerations:

Android

- By default, the `HAlign` attribute, which specifies the horizontal alignment of the text, is set to `left`.
- By default, the `TextColor` attribute, which specifies the text color, is set to `white`.
- For the `ClipType` attribute, Android devices support `CLIP_TYPE_CLIP`, which indicates that Android devices truncate the text if it is too long to display.

iOS

- By default, the `HAlign` attribute, which specifies the horizontal alignment of the text, is set to `center`.
- By default, the `TextColor` attribute, which specifies the text color, is set to `black`.
- For the `ClipType` attribute, iOS devices support `CLIP_TYPE_ELLIPSIS`, which indicates that iOS devices truncate the text that is too long to display and add an ellipsis to indicate the text has been truncated.

Example

This code sample displays plain text. Details on how the example code is rendered on various platforms follow the code sample.

```
view.add(new nUITextfieldElement(NUIID_MY_TEXTFIELD, "nUITextfieldElement"));
```

Platform	Platform-Specific Class and Visual Reference
Android	TextView "https://developer.android.com/reference/android/widget/TextView"
iOS	UILabel "https://developer.apple.com/design/human-interface-guidelines/ios/controls/labels/"

nUITimerObject

com.softwareag.mobile.runtime.nui.nUITimerObject

Use to add a timer object that waits a set period of time and then performs an automatic callback event after the time period elapses.

Usage Notes

- You can use the timer object to count up to a timestamp as a literal value. Set the `ActionTime` attribute to the required timestamp value and set the `Time` attribute using System.currentTimeMillis().

nUIViewDisplay

com.softwareag.mobile.runtime.nui.nUIViewDisplay

Use to create a view that contains other NativeUI objects. A view can contain any object except a nUIWindowDisplay object or another nUIViewDisplay object.

Usage Notes

- You can add a view (nUIViewDisplay) to a nUIWindowDisplay object.
- The following are platform-specific considerations:
  - The `InnerX` and `InnerY` attributes are set as follows:

Platform	InnerX Attribute Value	InnerY Attribute Value
Android	0 (zero)	0 (zero)
iOS	*screen_width* / 40	0 (zero)

- The following tables lists the number of nUINavbuttonElement objects that you can display in a nUIViewDisplay object based on platform:

Platform	nUINavbuttonElement objects allowed in a nUIViewDisplay object
Android	1-3
iOS	**iPhone:** 1-3 **iPad:** 1-6

Example

This code sample creates a view. Details on how the example code is rendered on various platforms follow the code sample.

```
nUIViewDisplay view = new nUIViewDisplay(NUIID_WEBVIEWELEMENT_VIEW);
view.setHeadertext("nUIViewDisplay");

nUINavbuttonElement ne = new nUINavbuttonElement(NUIID_BACK_BUTTON, "Back",
nUINavbuttonElement.TYPE_BACK, null);
view.add(ne);
```

Platform	Platform-Specific Class and Visual Reference
Android	View
iOS	UIView

nUIWebView

com.softwareag.mobile.runtime.nui.nUIWebView

Use to create a container for the nUIWebviewElement object, which is an object that allows for the display of rich web content. This object allows you to pass messages between the Java code and the JavaScript run time in the browser. For guidelines on how to approach this, see "JavaScript Bridge" on page 41.

Usage Notes

- The browser engines that mobile platforms use have differences in terms of how they display objects and support of JavaScript. You might need to develop platform-specific changes to support your content. Mobile Designer does not change your raw HTML content.
- Avoid invoking the Javascript alert() method from inside a nUIWebView object. For security reasons, some manufacturers configure their devices to block displaying

an alert dialog that results from a call to the Javascript alert() call. As an alternative, consider one of the following:

- Pass an event back to your Java code using a class that implements nUIWebviewCallBack, which in turn can then open a nUIAlertDialog object if an alert dialog is required.
- Use other web-based elements to display the alert information directly within your web page.

- There are two properties that react slightly differently depending on the platform.
  - `bgcolor`

    The `bgcolor` can be set at any time under iOS, and the web page will be redrawn to react to the change. Under Android, the changes to the `bgcolor` will only take affect when the webview redraws in response to changes in content (i.e., when the setURL() or setHTMLText() methods are called). For both platforms, the extent to which the web content is effected by this call is dependant on the rendering engine used in the browser itself, as well as the HTML content being displayed.
  - `Scaling web content to fit`

    This property is only usable on iOS by using UIWebView as browser engine. All other platforms ignore it. The setting will only affect page contents when the content of the page is changed (i.e. via setURL() or setHTMLText()).

  There is currently no support for altering `bgcolor`, changing the scaling of content to fit, or enabling/disabling overscrolling for Phoney.
- On iOS, you can switch from the default WKWebView engine to the deprecated UIWebView. You can set UIWebView as default browser engine for all nUIWebView and nUIWebviewElement instances by overriding the system property `ios.webview`. For more information about setting system properties, see *Using webMethods Mobile Designer*. Or you can pass this property directly into constructor as follows:

```
<code>
  Hashtable properties = new Hashtable();
  props.put("ios.webview", "UIWebview");
  nUIWebView webview = new nUIWebView(properties);
</code>
```

Example

This code sample creates a web view with navigation to the parent menu. Details on how the example code is rendered on various platforms follow the code sample.

```
protected nUIViewDisplay onCreateWebView()
{
  nUIWebView web_view = new nUIWebView(NUIID_WEB_VIEW);
  web_view.setHeadertext("nUIWebView");
  web_view.setURL("http://www.wikipedia.org/");
  nUINavbuttonElement ne = new nUINavbuttonElement(NUIID_BACK_TO_START_BUTTON,
"Back",
  nUINavbuttonElement.TYPE_BACK, null);
  web_view.add(ne);
  return web_view;
}
```

Platform	Platform-Specific Class and Visual Reference
Android	android.webkit.WebView
iOS	UIWebView "https://developer.apple.com/design/human-interface-guidelines/ios/views/web-views/"

nUIWebviewCallBack

Use to monitor when a user clicks a URL, and control the resulting action that the application takes in response to the user clicking the URL.

To use nUIWebviewCallBack, register a class that implements nUIWebviewCallBack with the web-based NativeUI object that contains the URL you want to monitor, for example, a nUIWebView or nUIWebviewElement object. When a user clicks a URL in the web-based NativeUI object, the application code can then take an appropriate action. For example, you might code the application to:

- Change the URL before passing it to the containing web-based NativeUI object to redirect the NativeUI object.
- Return a null to prevent a page load. This is useful when navigation to a new URL is not needed.
- Queue a new NativeUI event to allow the web-based NativeUI object to interact with the rest of the application.

nUIWebviewElement

com.softwareag.mobile.runtime.nui.nUIWebviewElement

Use to display rich web content from a local or remote source. This object allows you to pass messages between the Java code and the JavaScript run time in the browser. For guidelines on how to approach this, see "JavaScript Bridge" on page 41.

Usage Notes

- The browser engines that mobile platforms use have differences in terms of how they display objects and support of JavaScript. You might need to develop platform-specific changes to support your content. Mobile Designer does not change your raw HTML content.
- Avoid invoking the Javascript alert() method from inside a nUIWebviewElement object. For security reasons, some manufacturers configure their devices to block displaying

an alert dialog that results from a call to the Javascript alert() call. As an alternative, consider one of the following:

- Pass an event back to your Java code using a class that implements nUIWebviewCallBack, which in turn can then open a nUIAlertDialog object, if an alert dialog is required.
- Use other web-based elements to display the alert information directly within your web page.

- There are two properties that react slightly differently depending on the platform.
  - `bgcolor`

    The `bgcolor` can be set at any time under iOS, and the web page will be redrawn to react to the change. Under Android, the changes to the `bgcolor` will only take affect when the webview redraws in response to changes in content (i.e., when the setURL() or setHTMLText() methods are called). For both platforms, the extent to which the web content is effected by this call is dependant on the rendering engine used in the browser itself, as well as the HTML content being displayed.
  - `Scaling web content to fit`

    This property is only usable on iOS. All other platforms ignore it. The setting will only affect page contents when the content of the page is changed (i.e. via setURL() or setHTMLText()).

  There is currently no support for altering `bgcolor`, changing the scaling of content to fit or enabling/disabling overscrolling for Phoney.

Example

This code sample displays rich web content from a website. Details on how the example code is rendered on various platforms follow the code sample.

```
nUIWebviewElement webelement =
       new nUIWebviewElement(NUIID_MY_WEBVIEWELEMENT);
webelement.setHeight(250);
webelement.setURL("http://www.wikipedia.org/");
view.add(webelement);
```

Platform	Platform-Specific Class and Visual Reference
Android	android.webkit.WebView
iOS	UIWebView "https://developer.apple.com/design/human-interface-guidelines/ios/views/web-views/"

nUIWindowDisplay

com.softwareag.mobile.runtime.nui.nUIWindowDisplay

Use to add a window.

Usage Notes

- By default, the window has two panes, Pane 0 and Pane 1.
  - Use Pane 0 for the main pane. It occupies all available space.
  - Use Pane 1 for navigation using the nUINavView object.
- Panes are useful when supporting large mobile devices, such as tablets. For information about using panes, see "About the Native User Interface (NativeUI) Library" on page 10.
- You can add a nUIViewDisplay object to a nUIWindowDisplay object.
- Windows can also hold nUIViewDisplay objects as side views to provide a pop-up side menu throughout the application.