Implementing Dark Mode in iOS

Kurt Revis, UIKit Engineer
Tyler Fox, UIKit Engineer
Dark Mode is a new look
It’s easy to implement
Flexible and powerful
Dark Mode is a new look

It’s easy to implement

Flexible and powerful
Primary
Secondary
Tertiary
Quaternary
Secondary
Tertiary
Quaternary
Dark Mode

In iOS 13.0 and later, people can choose to adopt a dark system-wide appearance called Dark Mode. In Dark Mode, the system uses a darker color palette for all screens, views, menus, and controls, and it uses more vibrancy to make foreground content stand out against the darker backgrounds. Dark Mode supports all accessibility features.

People can choose Dark Mode as their default interface style, and they can use Settings to make their devices automatically switch to Dark Mode when ambient light is low.

Focus on your content. Dark Mode puts the focus on the content areas of your interface, allowing that content to stand out while the surrounding UI recedes into the background.

Test your design in both light and dark appearances. See how your interface looks in both appearances, and adjust your designs as needed to accommodate each one. Designs that work well in one appearance might not work in the other.

Ensure that your content remains comfortably legible in Dark Mode when you adjust the contrast and transparency accessibility settings. In Dark Mode, you should test your content with Increase Contrast and Reduce Transparency turned on, both separately and together. You may find places where dark text is less legible when it’s on a dark background. You might also find that turning on Increase Contrast in Dark Mode can result in reduced visual contrast between dark text and a dark background. Although people with strong vision might still be able to read lower contrast text, such text could be illegible for people with visual impairments. For guidance, see Color and Contrast.

Colors

The color palette in Dark Mode includes darker background colors and lighter foreground colors that are carefully selected to ensure contrast while maintaining a consistent feel.
Designing for Dark Mode

Use UIKit colors, materials, views, and controls

Customize colors and images when necessary
Dark Mode is a new look
It’s easy to implement
Flexible and powerful
Implementing Dark Mode

Using iOS 13 SDK implies Dark Mode support

You decide your app’s appearance
Use Dynamic Colors

Color ➔ white
Use Dynamic Colors

Color

light → white

dark → black
UIBlurEffect
style = .systemMaterial
UIBlurEffect
style = .systemMaterial

UIVisualEffectView
UIBlurEffect
style = .systemMaterial

UIVisualEffectView
UIBlurEffect
style = .systemMaterial

UIVisualEffectView
UIBlurEffect
style = .systemMaterial

UIVisualEffectView

UIVibrancyEffect
style = .fill
UIBlurEffect
style = .systemMaterial

UIVibrancyEffect
style = .fill

UIVisualEffectView
UIBlurEffect
style = .systemMaterial

UIVibrancyEffect
style = .fill

UIVisualEffectView
contentView

UIVisualEffectView
contentView
Demo
Large Title

Lorem ipsum dolor sit elit lamen, consectetur cillum adipiscing lucu, sed do eiusmod tempor incididunt ut labore et dolore magna aliquam. Ut enim ad minim veniam, quis nostrud exercitation ullamco tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Dark Mode is a new look
It’s easy to implement
Flexible and powerful
Dynamic Colors

- Color
  - light: white
  - dark: black
Large Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Resolving Dynamic Colors

UITraitCollection
userInterfaceStyle .dark

Dynamic Color ➔ black
Resolving Dynamic Colors

```swift
let dynamicColor = UIColor.systemBackground
let traitCollection = view.traitCollection
let resolvedColor = dynamicColor.resolvedColor(with: traitCollection)
```
Resolving Dynamic Colors

```swift
let dynamicColor = UIColor.systemBackground
let traitCollection = view.traitCollection
let resolvedColor = dynamicColor.resolvedColor(with: traitCollection)
```
Resolving Dynamic Colors

```swift
let dynamicColor = UIColor.systemBackground
let traitCollection = view.traitCollection
let resolvedColor = dynamicColor.resolvedColor(with: traitCollection)
```
let dynamicColor = UIColor.systemBackground
let traitCollection = view.traitCollection
let resolvedColor = dynamicColor.resolvedColor(with: traitCollection)
Custom Dynamic Colors

```swift
let dynamicColor = UIColor { (traitCollection: UITraitCollection) -> UIColor in
    if traitCollection.userInterfaceStyle == .dark {
        return .black
    } else {
        return .white
    }
}
```
let dynamicColor = UIColor { (traitCollection: UITraitCollection) -> UIColor in
    if traitCollection.userInterfaceStyle == .dark {
        return .black
    } else {
        return .white
    }
}
let dynamicColor = UIColor { (traitCollection: UITraitCollection) -> UIColor in
    if traitCollection.userInterfaceStyle == .dark {
        return .black
    } else {
        return .white
    }
}
let dynamicColor = UIColor { (traitCollection: UITraitCollection) -> UIColor in
    if traitCollection.userInterfaceStyle == .dark {
        return .black
    } else {
        return .white
    }
}
class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {
        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {
        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {
        // UIKit sets UITraitCollection.current to self.traitCollection
        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
Current Trait Collection

class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {
        // UIKit sets UITraitCollection.current to self.traitCollection
        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {
        // UIKit sets UITraitCollection.current to self.traitCollection
        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
Current Trait Collection

class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {
        // UIKit sets UITraitCollection.current to self.traitCollection
        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
## Current Trait Collection

<table>
<thead>
<tr>
<th>UIView</th>
<th>UIViewController</th>
<th>UIPresentationController</th>
</tr>
</thead>
<tbody>
<tr>
<td>draw()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Current Trait Collection

<table>
<thead>
<tr>
<th>UIView</th>
<th>UIViewController</th>
<th>UIPresentationController</th>
</tr>
</thead>
<tbody>
<tr>
<td>draw()</td>
<td>viewWillLayoutSubviews()</td>
<td>containerViewWillLayoutSubviews()</td>
</tr>
<tr>
<td>layoutSubviews()</td>
<td>viewDidLayoutSubviews()</td>
<td>containerViewDidLayoutSubviews()</td>
</tr>
</tbody>
</table>
# Current Trait Collection

<table>
<thead>
<tr>
<th>UIView</th>
<th>UIViewController</th>
<th>UIPresentationController</th>
</tr>
</thead>
<tbody>
<tr>
<td>draw()</td>
<td>varargin</td>
<td></td>
</tr>
<tr>
<td>layoutSubviews()</td>
<td>willLayoutSubviews()</td>
<td>containerViewWillLayoutSubviews()</td>
</tr>
<tr>
<td></td>
<td>viewDidLoad()</td>
<td>containerViewDidLayoutSubviews()</td>
</tr>
<tr>
<td>traitCollectionDidChange()</td>
<td>traitCollectionDidChange()</td>
<td>traitCollectionDidChange()</td>
</tr>
<tr>
<td>tintColorDidChange()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
let layer = CALayer()

layer.borderColor = UIColor.label.cgColor
let layer = CALayer()
let traitCollection = view.traitCollection

layer.borderColor = UIColor.label.cgColor
let layer = CALayer()

let traitCollection = view.traitCollection

layer.borderColor = UIColor.label.cgColor
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 1

let resolvedColor = UIColor.label.resolvedColor(with: traitCollection)
layer.borderColor = resolvedColor.cgColor
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 1
let resolvedColor = UIColor.label.resolvedColor(with: traitCollection)
layer.borderColor = resolvedColor.cgColor
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 2

traitCollection.performAsCurrent {
    layer.borderColor = UIColor.label.cgColor
}


let layer = CALayer()
let traitCollection = view.traitCollection

// Option 2

traitCollection.performAsCurrent {
    layer.borderColor = UIColor.label.cgColor
}

let layer = CALayer()
let traitCollection = view.traitCollection

// Option 3

UITraitCollection.current = traitCollection
layer.borderColor = UIColor.label.cgColor
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 3
UITraitCollection.current = traitCollection
layer.borderColor = UIColor.label.cgColor
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 3

let savedTraitCollection = UITraitCollection.current

UITraitCollection.current = traitCollection
layer.borderColor = UIColor.label.cgColor

UITraitCollection.current = savedTraitCollection
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 3

let savedTraitCollection = UITraitCollection.current
UITraitCollection.current = traitCollection
layer.borderColor = UIColor.label.cgColor
UITraitCollection.current = savedTraitCollection
When Dynamic Color Might Change

```swift
override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)
}
```
When Dynamic Color Might Change

```swift
override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)
}
```
override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)

    if traitCollection.hasDifferentColorAppearance(comparedTo: previousTraitCollection) {
        // Resolve dynamic colors again
    }
}
When Dynamic Color Might Change

```swift
override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)

    if traitCollection.hasDifferentColorAppearance(comparedTo: previousTraitCollection) {
        // Resolve dynamic colors again
    }
}
```
Dynamic Images

UIImageView

UITraitCollection
userInterfaceStyle .dark
Resolving Dynamic Images

```swift
let image = UIImage(named: "HeaderImage")
let asset = image?.imageAsset
let resolvedImage = asset?.image(with: traitCollection)
```
let image = UIImage(named: "HeaderImage")
let asset = image?.imageAsset
let resolvedImage = asset?.image(with: traitCollection)
let image = UIImage(named: "HeaderImage")
let asset = image?.imageAsset
let resolvedImage = asset?.image(with: traitCollection)
Resolving Dynamic Images

```swift
let image = UIImage(named: "HeaderImage")
let asset = image?.imageAsset
let resolvedImage = asset?.image(with: traitCollection)
```
Trait Collections
UIScreen

UIWindowScene

UIWindow

UIPresentationController

UIViewController
UIScreen

UIApplicationScene

UIApplication

UIViewController

UIPresentationController

UIWindow

UIWindowScene

UIView
UIScreen

UIWindowScene

UIWindow

UIPresentationController

UIViewController

UIView
let view = UIView()
let view = UIView()
let view = UIView()

traitCollection
let view = UIView()
addSubview(view)
addSubview(view)
Trait Collection Changes in iOS 13

Traits are predicted during initialization

traitCollectionDidChange(_:) called only for changes
# Trait Collection Changes in iOS 13

Enable debug logging with launch argument

```
-UITraitCollectionChangeLoggingEnabled YES
```

<table>
<thead>
<tr>
<th>Arguments Passed On Launch</th>
</tr>
</thead>
<tbody>
<tr>
<td>-UITraitCollectionChangeLoggingEnabled YES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environment Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

No Environment Variables
Using Trait Collections

Layout is the best time to use traits

- UIViewController.viewDidLoadWillLayoutSubviews()
- UIView.layoutSubviews()
- UIViewController.viewDidLoadDidLayoutSubviews()
Large Title

Lorem ipsum dolor sit er elit lamet, consectetur cillum adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Large Title

Lorem ipsum dolor sit er elit lamet, consectetur cillum adipiscing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Learn More
Overriding User Interface Style

class UIViewController {
    var overrideUserInterfaceStyle: UIUserInterfaceStyle
}

class UIView {
    var overrideUserInterfaceStyle: UIUserInterfaceStyle
}
Overriding User Interface Style

```swift
class UIViewController {
    var overrideUserInterfaceStyle: UIUserInterfaceStyle
}
```

```swift
class UIView {
    var overrideUserInterfaceStyle: UIUserInterfaceStyle
}
```

For entire app, set Info.plist key `UIUserInterfaceStyle` to `Light` or `Dark`
Overriding Traits

Existing API to override any traits

class UIPresentationController {
    var overrideTraitCollection: UITraitCollection?
}

class UIViewController {
    func setOverrideTraitCollection(_:UITraitCollection?, forChild: UIViewController)
}

Only specify values for traits you want to override
Dark Mode API Updates
Status Bar
Before iOS 13

.default

.lightContent
Status Bar

iOS 13
Status Bar

iOS 13

- .default
  - light
    - .darkContent
  - dark
    - .lightContent
UIActivityIndicatorView

Deprecated styles

.gray

.white

.whiteLarge
UIActivityIndicatorView

New dynamic styles `.medium` and `.large`

Use the `color` property to set your own
Drawing Text

This is a label

This is an editable text field

This is a text view. Lorem ipsum dolor sit er elit
l amet, consectetur cillum adipiscing pecu, sed
do eiusmod tempor incididunt ut labore et dolore
magna aliqua. Ut enim ad minim veniam, quis
n nostrud exercitation ullamco laboris nisi ut aliquip
ex ea commodo consequat. Duis aute idure dolor
in reprehenderit in voluptate velit esse cillum
dolore eu fugiat nulla pariatur. Excepteur sint
occucaecat.
Drawing Attributed Text

```swift
let attributes: [NSAttributedString.Key: Any] = [
    .font: UIFont.systemFont(ofSize: 36.0)
]
```

This text is an attributed string.
Drawing Attributed Text

```swift
let attributes: [NSAttributedString.Key: Any] = [
    .font: UIFont.systemFont(ofSize: 36.0)
    .foregroundColor: UIColor.label
]
```

This text is an attributed string.
let attributes: [NSAttributedString.Key: Any] = [
    .font: UIFont.systemFont(ofSize: 36.0)
    .foregroundColor: UIColor.label
]

This text is an attributed string.
Must opt in to dark mode

Declare supported color schemes with `color-scheme`

Use `prefers-color-scheme` media query for custom colors and images
tvOS

Apps built using tvOS 13 SDK are expected to support dark mode

Most new API is available
iPad Apps for Mac

Same API

Follows system setting

Matches AppKit colors and materials
Apps on iOS 13 are expected to support dark mode
Apps on iOS 13 are expected to support dark mode
Use system colors and materials
Apps on iOS 13 are expected to support dark mode
Use system colors and materials
Create your own dynamic colors and images
Apps on iOS 13 are expected to support dark mode
Use system colors and materials
Create your own dynamic colors and images
Leverage flexible infrastructure
More Information

developer.apple.com/wwdc19/214