iOS Image Overlay Swift

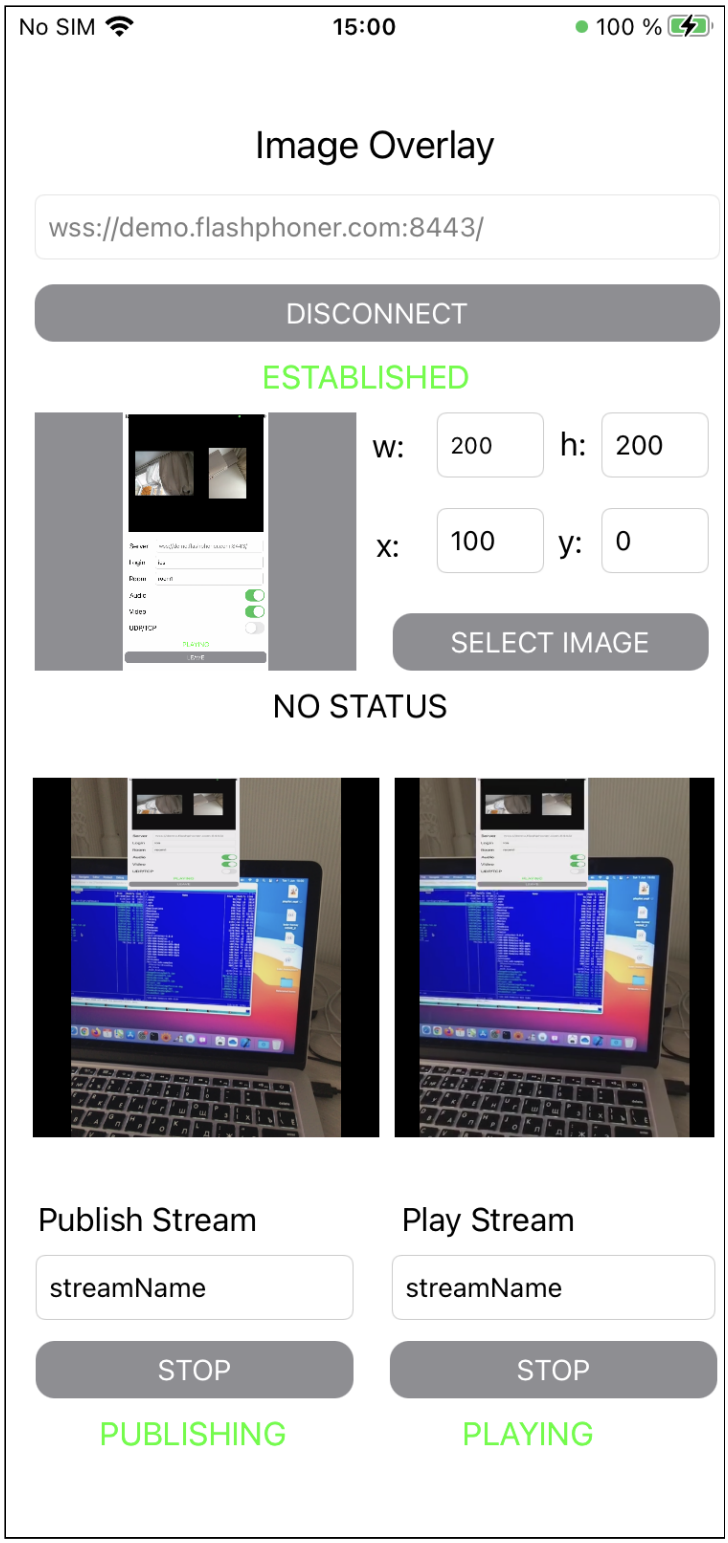
A video published zooming and picture overlaying example

The example shows how to zoom video published (zoom in and zoom out by pinching) and how to overlay a picture from device gallery.

On the screensot below, the picture is added to video to the position defined with resizing.

Inputs:

- `WCS URL` - WCS server address
- `w` - picture overlaing width
- `h` - picture overlaying height
- `x` - picture top left corner position by horizontal axis
- `y` - picture top left corner position by vertical axis
- `Select image` - picture choose from gallety button



The picture, its size and placement can be changed during stream publishing on the fly.

Analyzing the example code

To analyze the example code take ImageOverlaySwift example version which is available on GitHub:

- `ImageOverlayViewController` - main application view class (implementation file ImageOverlayViewController.swift)
- `CameraVideoCapturer` - class to implement video capturing and handling (implementation file CameraVideoCapturer.swift)

1. API import

code

```
import FPWCSApi2Swift
```

2. Video capturer initialization

code

```
var capturer: CameraVideoCapturer = CameraVideoCapturer()
```

3. Session creation and connecting to the server

`WCSSession`, `WCSSession.connect` code

The following session parameter are set:

- WCS server URL
- server backend REST hook application name `defaultApp`

```
@IBAction func connectPressed(_ sender: Any) {
    changeViewState(connectButton, false)
    if (connectButton.title(for: .normal) == "CONNECT") {
        if (session == nil) {
            let options = FPWCSApi2SessionOptions()
            options.urlServer = urlField.text
            options.appKey = "defaultApp"
            do {
                try session = WCSSession(options)
            } catch {
                print(error)
            }
        }
        ...
        changeViewState(urlField, false)
        session?.connect()
    } else {
        session?.disconnect()
    }

}
```

4. Stream publishing

`WCSSession.createStream`, `WCSStream.publish` code

The following parameters are passed to `createStream` method:

- stream name to publish
- local video view
- video capturer object

```
@IBAction func publishPressed(_ sender: Any) {
    changeViewState(publishButton,false)
    if (publishButton.title(for: .normal) == "PUBLISH") {
        let options = FPWCSApi2StreamOptions()
        options.name = publishName.text
        options.display = localDisplay.videoView
        options.constraints = FPWCSApi2MediaConstraints(audio: true,
videoCapturer: capturer);
        do {
        publishStream = try session!.createStream(options)
        } catch {
            print(error);
        }
        ...
        do {
            try publishStream?.publish()
            capturer.startCapture()
        } catch {
            print(error);
        }
    } else {
        do {
            try publishStream?.stop();
        } catch {
            print(error);
        }

    }

}
```

5. Stream playback

`WCSSession.createStream`, `WCSStream.play` code

The following parameters are passed to createStream method:

- stream name to play
- remote video view

```
@IBAction func playPressed(_ sender: Any) {
    changeViewState(playButton,false)
    if (playButton.title(for: .normal) == "PLAY") {
        let options = FPWCSApi2StreamOptions()
        options.name = playName.text;
        options.display = remoteDisplay.videoView;
        do {
        playStream = try session!.createStream(options)
        } catch {
            print(error)
        }
        ...
        do {
            try playStream?.play()
        } catch {
            print(error);
        }
    } else{
        do {
            try playStream?.stop();
        } catch {
            print(error);
        }
    }
}
```

6. Stop stream playback

`WCSStream.stop` code

```
@IBAction func playPressed(_ sender: Any) {
    changeViewState(playButton,false)
    if (playButton.title(for: .normal) == "PLAY") {
        ...
    } else{
        do {
            try playStream?.stop();
        } catch {
            print(error);
        }
    }
}
```

7. Stop stream publishing

`WCSStream.stop` code

```
@IBAction func publishPressed(_ sender: Any) {
    changeViewState(publishButton,false)
    if (publishButton.title(for: .normal) == "PUBLISH") {
        ...
    } else {
        do {
```

```
            try publishStream?.stop();
        } catch {
            print(error);
        }

    }

}
```

8. Zooming function call by pinching local voideo view

code

```
@IBAction func pinchOnLocalDisplay(_ sender: UIPinchGestureRecognizer) {
    if sender.state == .changed {
        self.capturer.scale(velocity: sender.velocity)
    }
}
```

9. Picture choosing from gallery and claaing function to update image to overlay

code

```
@IBAction func selectImagePressed(_ sender: Any) {
    imagePicker.allowsEditing = false
    imagePicker.sourceType = .photoLibrary
    DispatchQueue.main.async {
        self.present(self.imagePicker, animated: true, completion: nil)
    }
}

func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any])
{
    guard let image = info[.originalImage] as? UIImage else {
        return;
    }

    selectedImage = image
    imageView.image = selectedImage
    updateOverlayImage()

    DispatchQueue.main.async {
        picker.dismiss(animated: true, completion: nil)
    }
}
```

10. Image resizing, position calculation and function calling to update image to overlay

code

```
func updateOverlayImage() {
    if let selectedImage = selectedImage {
        let resizeImage = resize((selectedImage.cgImage)!,
selectedImage.imageOrientation)

        let overlayImage = CIImage.init(cgImage: (resizeImage)!)
        let overX = CGFloat(Int(overlayX.text ?? "0") ?? 0)
        let overY = CGFloat(Int(overlayY.text ?? "0") ?? 0)
        let movedImage = overlayImage.oriented(.left).transformed(by:
CGAffineTransform(translationX: overY, y: overX))
        capturer.updateOverlayImage(movedImage)
    } else {
        capturer.overlayImage = nil
        return
    }
}
```

11. Video scaling implementation

code

```
func scale(velocity: CGFloat) {
    guard let device = self.device else { return }

    let maxZoomFactor = device.activeFormat.videoMaxZoomFactor
    let pinchVelocityDividerFactor: CGFloat = 15


    do {
        try device.lockForConfiguration()
        defer { device.unlockForConfiguration() }

        let desiredZoomFactor = device.videoZoomFactor + atan2(velocity,
pinchVelocityDividerFactor)
        device.videoZoomFactor = max(1.0, min(desiredZoomFactor,
maxZoomFactor))
    } catch {
        print(error)
    }
}
```

12. Image overlaying implementation

code

```
let pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer)
...
if (overlayImage != nil) {
    let inputImage = CIImage.init(cvImageBuffer: pixelBuffer!);
    let combinedFilter = CIFilter(name: "CISourceOverCompositing")!
```

```
    combinedFilter.setValue(inputImage, forKey: "inputBackgroundImage")
    combinedFilter.setValue(overlayImage, forKey: "inputImage")

    let outputImage = combinedFilter.outputImage!
    let tmpcontext = CIContext(options: nil)
    tmpcontext.render(outputImage, to: pixelBuffer!, bounds:
outputImage.extent, colorSpace: CGColorSpaceCreateDeviceRGB())

}
```