INFOnline
Integration Guide

Plattform: iOS
SZM-Library Version: 1.2.0

INFOnline GmbH
Brühler Straße 9
53119 Bonn

Tel.: +49 (0) 228 / 410 29 - 0
Fax: +49 (0) 228 / 410 29 - 66

www.INFOnline.de
info@INFOnline.de

Inhalt

1 Über dieses Dokument

Das vorliegende Dokument dient dazu, alle notwendigen Informationen für die technische Integration und Nutzung der SZM-Library in Apps mit dem Betriebssystem iOS zur Verfügung zu stellen. **Die unterstützte App-Plattfom** ist hier **iOS ab Version 7.0**.

Es ist in 5 Kapitel gegliedert:

1. Kapitel „Über dieses Dokument“ ermöglicht einen Überblick zum Aufbau und zur Zielsetzung dieses Integration Guides.
2. Kapitel „INFOnline SZM-Library (iOS)“ erläutert die Rand- und Rahmenbedingungen des Messinstruments.
3. Kapitel „Integration der SZM-Library iOS“ liefert die technischen Informationen für den Einbau des Messinstruments.
4. Kapitel „Vorgaben zum Aufruf der Library“ geht auf Vorgaben zur Verwendung der Messlibrary im Kontext der Messung SZM Mobile Applications ein.
5. Falls Sie Fragen oder Anregungen haben, kontaktieren Sie uns einfach. Alle Kontaktdaten finden Sie im Schlusskapitel „Kontakt“.

2 INFOnline SZM-Library (iOS)

Die INFOnline SZM-Library für Android (im Folgenden auch „IOLib“ genannt) ist eine Software-Bibliothek, welche die Nutzung von Android Apps erfasst, speichert und zum Zwecke der Validierung und Kontrolle an ein geeignetes Backend versendet. INFOnline stellt dieses Backend bereit.

2.1 Bereitstellung

Die IOLib iOS wird von INFOnline zum Download zur Verfügung gestellt. Die Zugangsdaten erhalten Sie per E-Mail.

Im Download Bereich befinden sich

- die Release Notes als Textdatei
- das Change Log als Textdatei
- ein Verzeichnis **INFOnlineLibrary.framework**: Die IOLib wird als „Framework“ bereitgestellt und kann leicht in bereits existierende iOS-Projekte integriert werden
- ein Verzeichnis **INFOnlineLibrarySample**: Eine Beispiel-App mit integrierter IOLib
- ein Verzeichnis **INFOnlineLibrarySample+WatchKit**: Ein Beispiel-Projekt, welches den Einsatz der IOLibrary für iOS mit einer WatchKit Extension (nur WatchOS 1) demonstriert

2.2 Anforderungen

Die SZM-Library für iOS unterstützt ausschließlich die Integration über die Entwicklungsumgebung Xcode unter MacOSX.

2.2.1 Entwicklungsumgebung

Um die Integration der IOLib iOS durchzuführen, sind folgende Voraussetzungen notwendig:

- MacOSX 10.10 (Yosemite) und höher
- iPhone SDK 9.1 und höher
- Xcode 7.0 und höher
- Objective-C oder Swift

iOS APPs, welche die IOLib iOS benutzen, müssen min. mit iOS SDK 8.4 kompiliert werden.

Deployment Target muss mind. auf 7.0 eingestellt werden.

2.2.2 Betriebssystem/Plattform

Die IOLib iOS unterstützt den Betrieb unter 32-Bit und 64-Bit Architekturen.
Die IOLib iOS setzt für den Betrieb iOS 7.0 oder höher voraus.

2.2.3 Kompatibilität

Eine neue Version von Xcode bzw. des iOS-Betriebssystems macht evtl. ein Update der SZM-Library notwendig. Eine vollständige Aufwärtskompatibilität kann u.U. nicht gewährleistet werden.

2.3 Funktionsweise

2.3.1 Wann muss die SZM-Library aufgerufen werden ?

Der Aufruf der Funktionen der IOLib innerhalb der App ist an bestimmte Events gebunden. Hierzu werden für die App-Anbieter seitens der AGOF und IVW Vorgaben bzw. Empfehlungen formuliert, an welchen Stellen der App bzw. bei welchen Nutzer-Aktionen die SZM-Library aufgerufen werden soll und welche Informationen zu übermitteln sind.

Die Vorgaben zum Aufruf der IOLib innerhalb der App werden in Kapitel 4 (*Vorgaben zum Aufruf der SZM-Library*) beschrieben.

2.3.2 Offline-Nutzung

Die IOLib iOS unterstützt die Nutzung mobiler Apps ohne aktive Internet-Verbindung. Die Events der Offline-Nutzung werden gespeichert und bei nächster Gelegenheit (sobald eine Internet-Verbindung besteht) an das Mess-Backend übertragen. Pro Event werden ein Timestamp, der jeweilige InternetConnection-Status sowie weitere Daten erfasst und dokumentieren damit den Zeitpunkt und die Rahmenbedingungen der Offline-Nutzung.

2.3.3 Übertragung der Messdaten

Um die Datenübertragung möglichst effizient zu gestalten und die Offline-Nutzung zu ermöglichen, werden die Messdaten nicht direkt synchron zum Zeitpunkt der Messung an das Backend übertragen, sondern in einer „Messdaten-Queue“ gesammelt.

Der zu übertragende Datensatz wird kontinuierlich mit den neuen Messdaten konkateniert, sobald ein bestimmtes Größenlimit erreicht ist, wird ein neuer Datensatz gebildet und der vorherige Datensatz zum Versand freigegeben.

Der Versand der Daten selbst findet asynchron statt. Dadurch wird die Interaktion des Benutzers mit der App nicht verzögert oder blockiert.

2.3.4 Opt-out-Funktion und Datenschutzerklärung

Die User einer App müssen informiert werden, dass die App die Nutzeraktionen misst und an das Mess-System der INFOnline überträgt. Für diesen Zweck stellt INFOnline eine Datenschutzerklärung zur Verfügung, die unter https://www.infonline.de/downloads/ abgerufen werden kann. Bitte binden Sie diesen Text an entsprechender Stelle in die App ein.

Darüberhinaus muss dem Nutzer eine Opt-out-Funktion gegeben werden. Die Implementierung dieser obliegt dem Entwickler der jeweiligen App. Nach Einbindung der Funktion können die Nutzer der App das Opt-out aktivieren und deaktivieren. Sofern das Opt-out aktiviert wird, wird kein Zählimpuls ausgelöst.

Die technischen Details zur Einbindung der Opt-Out-Funktion sind in Kapitel 3.2.5 (*Einbindung Opt-out-Funktion*) aufgeführt.

3 Integration der SZM-Library iOS

Dieses Kapitel beschreibt die technische Integration der SZM-Library in eine iOS-Projektstruktur.

3.1 Thread-safe

Die IOLib für iOS ist nicht Thread-safe. Alle Aufrufe von Methoden der Library müssen im Main Thread erfolgen.

3.2 IOLib Dateien

Die SZM-Library für iOS umfasst folgende Dateien

- **RELEASE_NOTES.txt**

  Diese Datei enthält Informationen zum Release der IOLib.

- **CHANGE_LOG.txt**

  Diese Datei enthält Informationen zur Änderungshistorie der IOLib.

- **INFOnlineLibrary.framework**

  Die IOLibrary zur Erfassung der Nutzungsdaten einer iOS-App

- **INFOnlineLibrarySample**

  Ein Beispiel-Projekt, welches den Einsatz der IOLibrary für iOS demonstriert

- **INFOnlineLibrarySample+WatchKit**

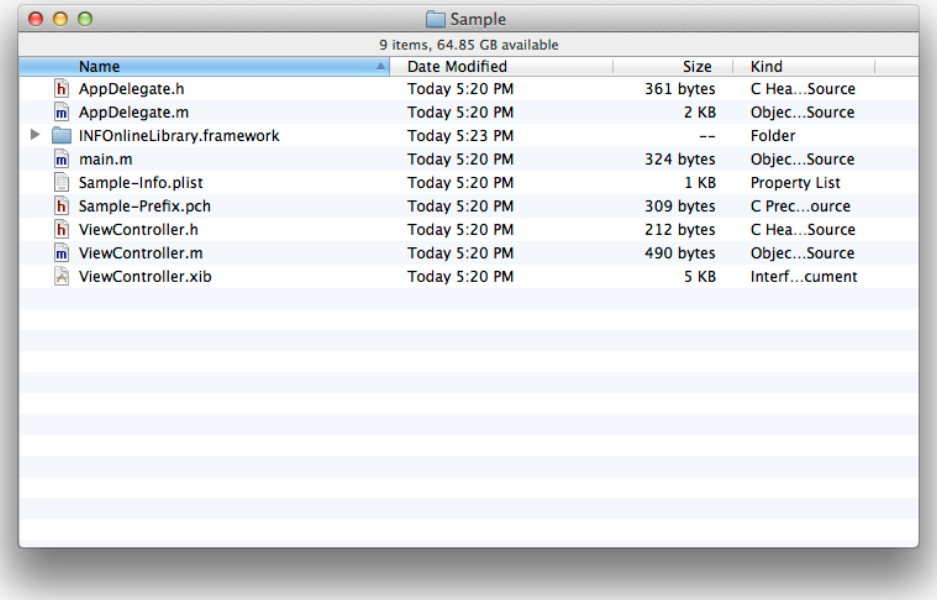
  Ein Beispiel-Projekt, welches den Einsatz der IOLibrary für iOS mit einer WatchKit Extension (nur WatchOS 1) demonstriert

3.3 Integration des IOLib iOS Frameworks

Die Integration erfolgt in wenigen einfachen Schritten.

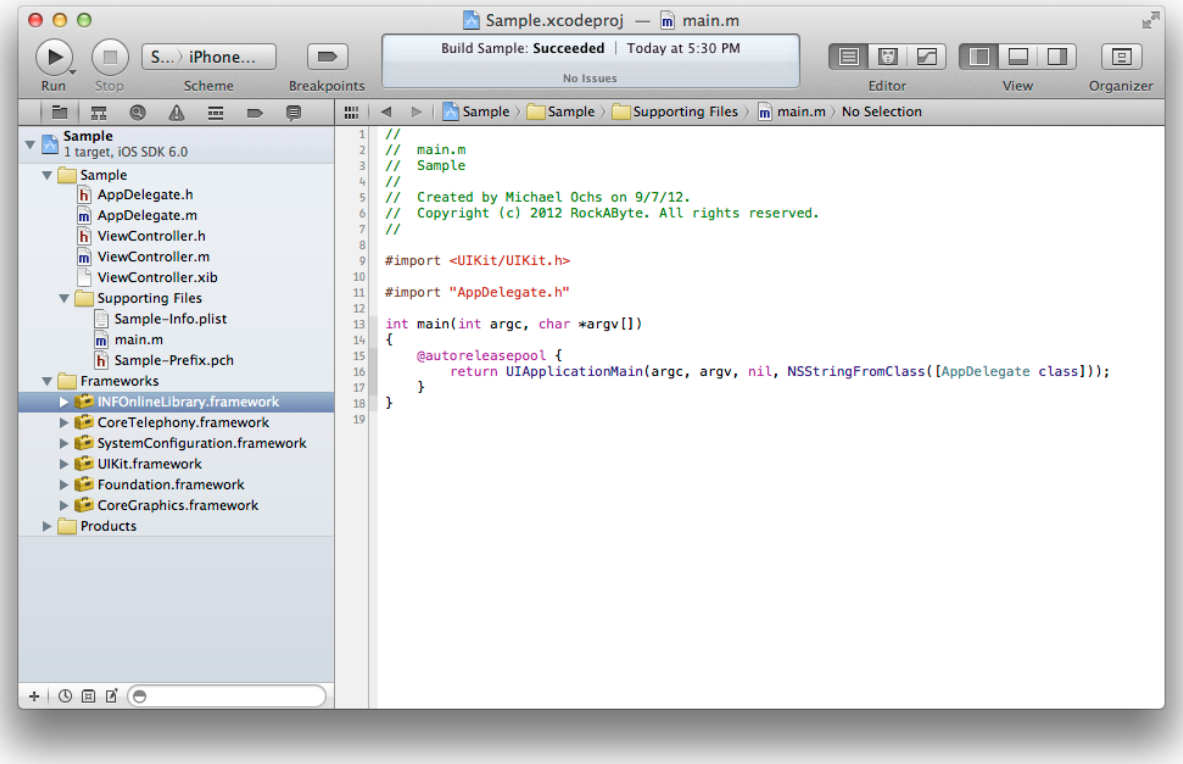
1. **Im Finder:**

Den Ordner „INFOnlineLibrary.framework“ in den Projekt-Ordner kopieren (oder per Drag’n’Drop ziehen)



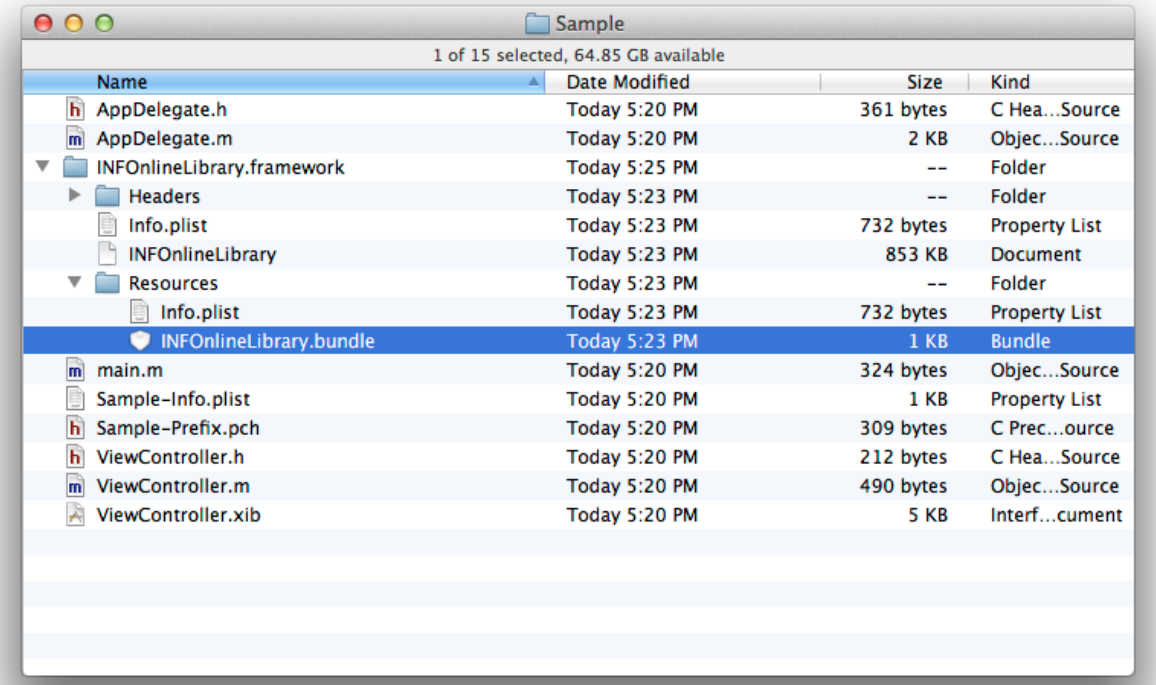
2. In Xcode:

Die Datei „INFOnlineLibrary.framework“ dem Projekt hinzufügen



3. **In Xcode**:

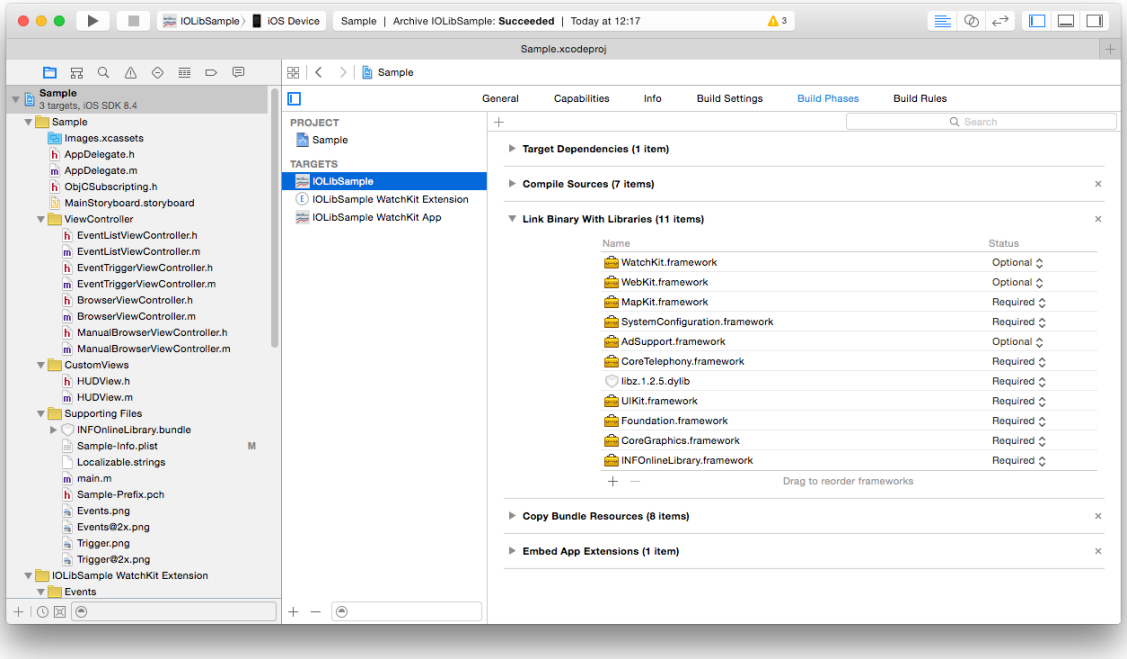
Die Datei „INFOnlineLibrary.bundle“ aus dem im Projekt befindlichen Order „INFOnlineLibrary.framework/Resources“ zum App-Projekt hinzufügen.



4. In Xcode:

Linking der Frameworks **CoreTelephony**, **SystemConfiguration** sowie **AdSupport**

WATCHKIT: Linking der Frameworks **WatchKit** und **WebKit**.



HINWEIS

Um eine korrekte und vollständige Messung für die AGOF Studienteilnahme zu gewährleisten, müssen die o.g. Frameworks eingebunden werden.
Wenn Sie die Frameworks nicht einbinden, kann der für die AGOF-Messung erforderliche Identifier (Advertising Identifier / IDFA) nicht übermittelt werden - eine Teilnahme an der AGOF-Messung ist somit nicht möglich.

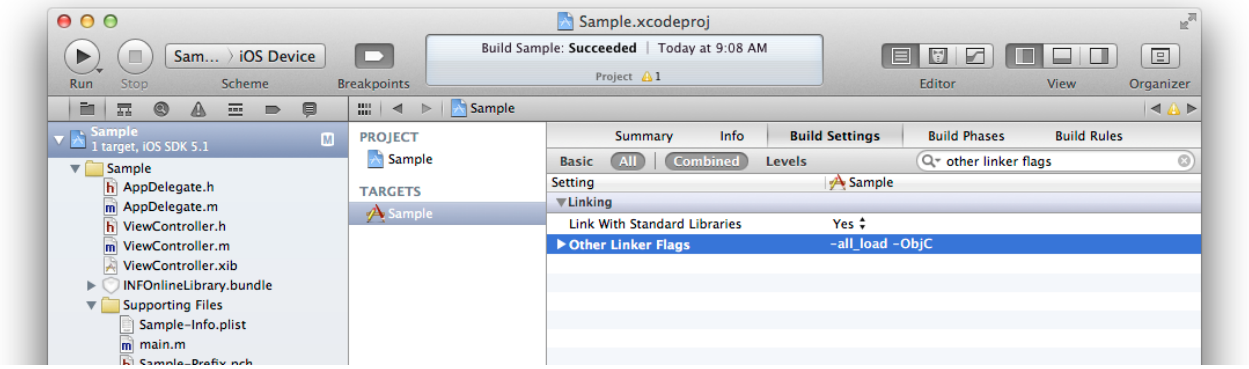
Falls Sie in der Applikation keine Werbeplätze anbieten und aufgrund der Bestimmungen der Apple iTunes Connect Developer Guidelines eine Nutzung der Advertising Identifier nicht möglich ist, wenden Sie sich bitte an das Service & Support Team der INFOnline unter der Rufnummer 0228 / 410 29 77 oder per E-Mail an support@INFOnline.de.

Sie haben die technische Möglichkeit, anstelle der Advertising ID den Vendor Identifier (IDVA) zu nutzen. Hierzu entfernen Sie bitte das Linking des AdSupport-Frameworks. Als Konsequenz daraus erhebt die IOLib dann den Vendor Identifier (IDVA) anstelle der Advertising Identifier (IDFA).

HINWEIS Wir empfehlen Ihnen stets die Umsetzung mit dem Advertising Identifier (IDFA) gemäß der Dokumentation. Eine Teilnahme bei der AGOF ist ohne die Verwendung des Advertising Identifiers nicht möglich!

5. In Xcode: Build Settings

Überprüfung der Einstellung „other linker flags“: Hier muss „-ObjC“ eingetragen sein.



6. In Xcode: main.m

HINWEIS Als Basisklasse der App darf nicht UIApplication, sondern es muss IOLApplication zum Start der Anwendung verwendet werden!

Objektive-C:

```
int main(int argc, char *argv[]) {

    @autoreleasepool {
        return UIApplicationMain(argc,
                                 argv,
                                 @"IOLApplication",
                                 NSStringFromClass([AppDelegate class]));
    }
}
```

Swift:

```
UIApplicationMain(Process.argc,
                  Process.unsafeArgv,
                  NSStringFromClass(IOLApplication),
                  NSStringFromClass(AppDelegate))
```

7. In Xcode:

Objective-C:

Import der IOLib Header im ApplicationDelegate und in den ViewControllern (alternativ im Prefix Header)

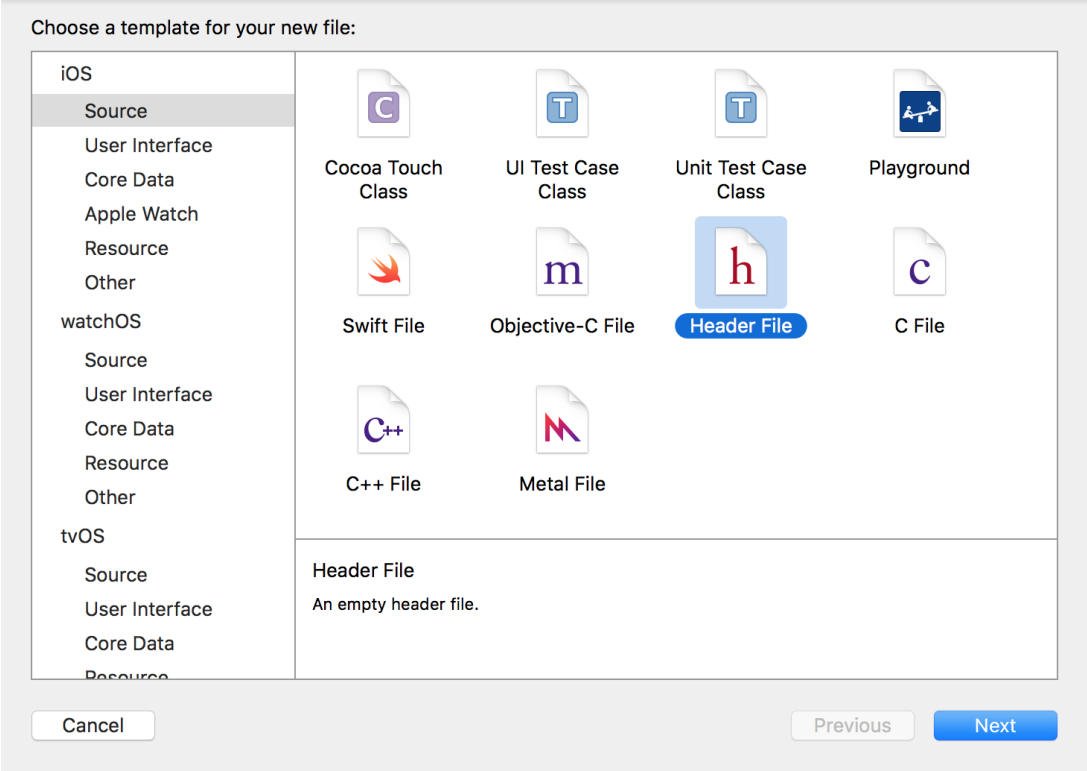
```
#import <INFOnlineLibrary/INFOnlineLibrary.h>
```

HINWEIS Es sollte immer der Umbrella Header benutzt werden und niemals einzelne Header aus dem Framework!

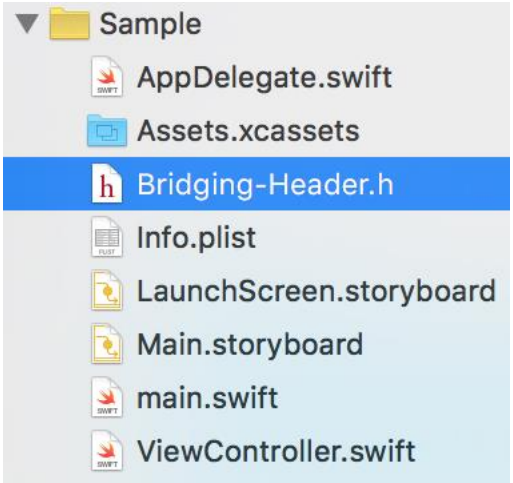
Swift:

Für die Verwendung in einem Swift-Projekt muss ein sog. Bridging Header angelegt werden:

1. Neuen Objective-C Header hinzufügen (im Beispiel „Bridging-Header“ genannt):



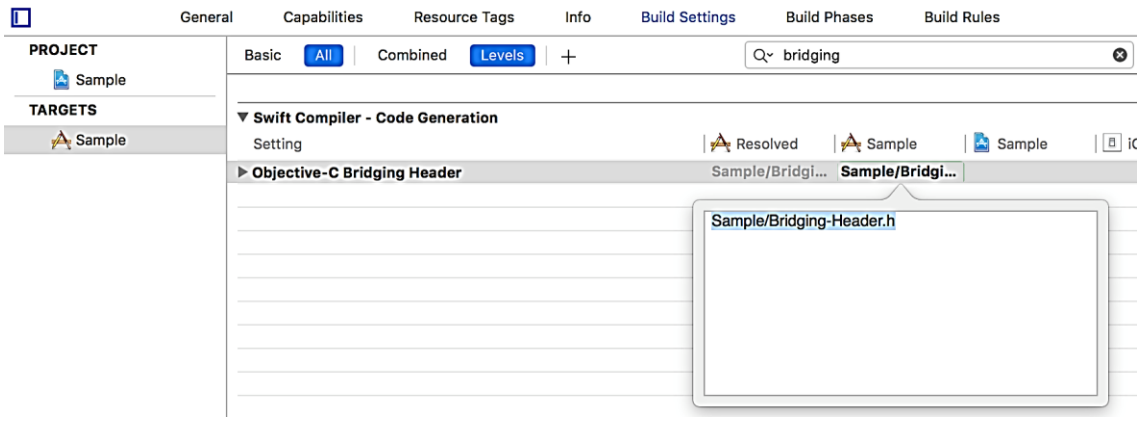
2. Den Bridging Header im Projektbaum wählen:

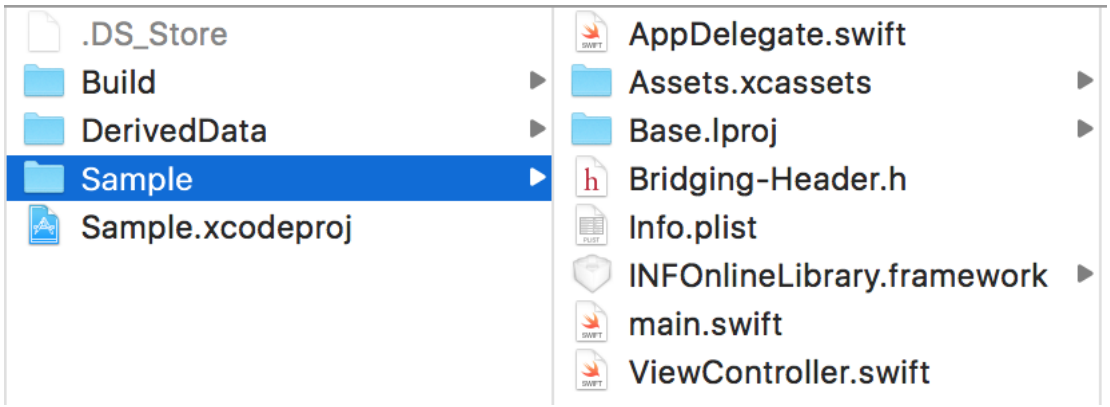


3. Den Bridging Header mit folgendem Inhalt füllen:

```
//
//  Bridging-Header.h
//  Sample
//

#ifndef Bridging_Header_h
#define Bridging_Header_h

#import <INFOnlineLibrary/INFOnlineLibrary.h>

#endif
```

4. Bei den Projekteinstellungen zum gewünschten Target wechseln und in „Build Settings“ nach „bridging“ suchen. Unter „Objective-C Bridging Header“ den relativen Pfad zur Header-Datei hinterlegen (im Beispiel findet sich die Datei im Unterordner „Sample“):





HINWEIS Es sollte immer der Umbrella Header benutzt werden und niemals einzelne Header aus dem Framework!

8. KIT: In XCode: ApplicationDelegate / AppDelegate

Für die Verwendung einer WatchKit Extension muss der **ApplicationDelegate** / **AppDelegate** von der Klasse **IOLAppDelegate** erben. Um eine spätere Ergänzung einer WatchKit Extension zu vereinfachen, empfiehlt es sich, die **IOLAppDelegate** von vorneherein als Basisklasse für den **ApplicationDelegate** / **AppDelegate** zu verwenden.

```
#import <INFOnlineLibrary/INFOnlineLibrary.h>

@interface AppDelegate : IOLAppDelegate

@end
```

Objective-C:

```
//
//  AppDelegate.h
//  Sample
//
//  Created by Michael Ochs on 10/1/12.
//  Copyright (c) 2012 RockAByte GmbH. All rights reserved.
//

#import <INFOnlineLibrary/INFOnlineLibrary.h>

@interface AppDelegate : IOLAppDelegate

@end
```

Swift:

```
//
//  AppDelegate.swift
//  Sample
//

class AppDelegate: IOLAppDelegate
```

9. In XCode:

Initialisierung und Start einer Session der IOLib beim Application-Start:

HINWEIS: Für die Verwendung einer WatchKit Extension muss die Methode startSession implementiert werden und von dort die Session gestartet werden. Um eine spätere Ergänzung einer WatchKit Extension zu vereinfachen, empfiehlt es sich, die Session von vorneherein in dieser Methode zu starten.

Objective-C:

```
@implementation AppDelegate

- (void)applicationDidFinishLaunching:(UIApplication*)application {

  [self startSession];

   // Other code
}

- (void)startSession {

  // Initialisierung der IOLib; Session-Start
  [[IOLSession defaultSession] startSessionWithOfferIdentifier:@"<ANGEBOTSKENNUNG>"];
}
```

```
1  //
2  //  AppDelegate.m
3  //  Sample
4  //
5  //  Created by Michael Ochs on 10/1/12.
6  //  Copyright (c) 2012 RockAByte GmbH. All rights reserved.
7  //
8
9  #import "AppDelegate.h"
10
11
12 @implementation AppDelegate
13
14 - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
15 {
16     [self startSession];
17
18     return YES;
19 }
20
21 - (void)startSession {
22
23     BOOL isOptOut = [[NSUserDefaults standardUserDefaults] boolForKey:@"IOLOptOut"];
24     if (!isOptOut) {
25         [[IOLSession defaultSession] startSessionWithOfferIdentifier:@"iamtest"];
26     }
27     [[IOLSession defaultSession] setDebugLogLevel:IOLDebugLevelVerbose];
28 }
```

Swift:

```
override func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject : AnyObject]?) -> Bool {
  self.startSession()

    // Other code
}

Override func startSession() {

  // Initialisierung der IOLib; Session-Start
  IOLSession.defaultSession().startSessionWithOfferIdentifier("<ANGEBOTSKENNUNG>")
}
```

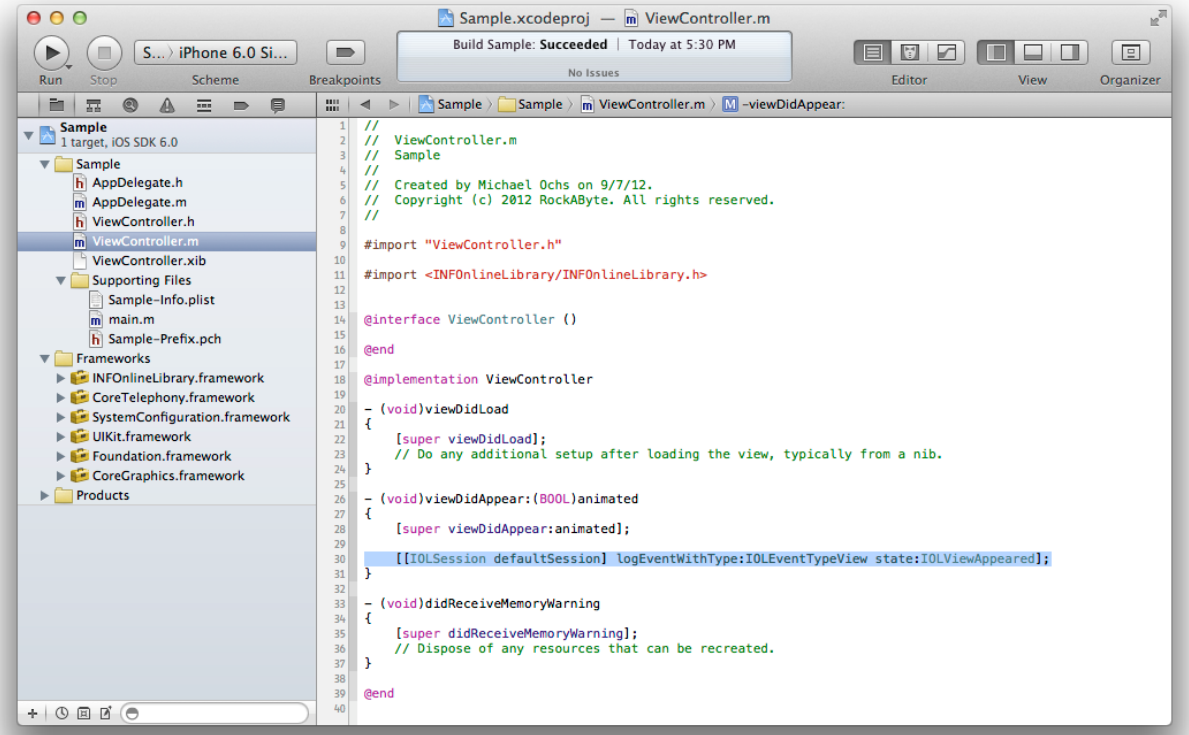
```
//
//  AppDelegate.swift
//  Sample
//

import UIKit

class AppDelegate: IOLAppDelegate {

    override func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject : AnyObject]?) -> Bool {

        self.startSession()

        return true
    }

    override func startSession() {

        let isOptOut = NSUserDefaults.standardUserDefaults().boolForKey("IOLOptOut")
        if (!isOptOut) {
            IOLSession.defaultSession().startSessionWithOfferIdentifier("iamtest")
        }

        IOLSession.defaultSession().debugLogLevel = 4 //IOLDebugLevelVerbose
    }
}
```

10. In Xcode:

Events können in den View Controllern der App geloggt werden, z.B. der Aufruf eines Views:

Objective-C:

```
// Tracking View Appeared
   [[IOLSession defaultSession] logEventWithType:IOLEventTypeView
                                           state:IOLViewAppeared];
```



Swift:

```
// Tracking View Appeared
    IOLSession.defaultSession().logEventWithType(IOLEventType.View,
                                                 state: IOLViewAppeared)
```

```
13      override func viewDidAppear(animated: Bool) {
14
15          super.viewDidAppear(animated)
16
17          IOLSession.defaultSession().logEventWithType(IOLEventType.View, state: IOLViewAppeared)
18      }
```

3.4 SZM-Library Funktionen

Die SZM-Library für iOS bietet die im Folgenden beschriebenen Funktionen:

3.4.1 Start einer Session

- **(void)startSessionWithOfferIdentifier:(NSString*)offerID**;

HINWEIS: **Die IOLib muss vor der Erfassung der Events gestartet werden.** Dabei muss die Angebotskennung der App als Parameter übergeben werden.

Parameter:

- **Angebotskennung (mandatory)**

Die eindeutige Kennung des Angebots der jeweiligen App. Die Angebotskennung wird von INFOnline pro App und pro Betriebssystem eindeutig vergeben.

Beispiel (hier ist die Angebotskennung “iamtest”):

Objective C:

```
//
//  AppDelegate.m
//  Sample
//
//  Created by Michael Ochs on 10/1/12.
//  Copyright (c) 2012 RockAByte GmbH. All rights reserved.
//

#import "AppDelegate.h"


@implementation AppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    [self startSession];

    return YES;
}

- (void)startSession {

    BOOL isOptOut = [[NSUserDefaults standardUserDefaults] boolForKey:@"IOLOptOut"];
    if (!isOptOut) {
        [[IOLSession defaultSession] startSessionWithOfferIdentifier:@"iamtest"];
    }
    [[IOLSession defaultSession] setDebugLogLevel:IOLDebugLevelVerbose];
}
```

Swift:

```swift
//
//  AppDelegate.swift
//  Sample
//

import UIKit

class AppDelegate: IOLAppDelegate {

    override func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject : AnyObject]?) -> Bool {

        self.startSession()

        return true
    }

    override func startSession() {

        let isOptOut = NSUserDefaults.standardUserDefaults().boolForKey("IOLOptOut")
        if (!isOptOut) {
            IOLSession.defaultSession().startSessionWithOfferIdentifier("iamtest")
        }

        IOLSession.defaultSession().debugLogLevel = 4 //IOLDebugLevelVerbose
    }
}
```

3.4.2 Logging eines Events

Die Messdaten werden mittels des Aufrufs **logEventWithType** erfasst. Dabei können bis zu drei Parameter übergeben werden, zwei davon sind optional.

- **(void)logEventWithType:(IOLEventType)type state:(IOLEventState)state;**

- **(void)logEventWithType:(IOLEventType)type state:(IOLEventState)state**
  category:(NSString*)category
  comment:(NSString*)comment;

Der erste Aufruf ist eine Convenience-Funktion, welche intern die Letztgenannte aufruft.

Die fehlenden Werte werden dann um **nil** bzw. Default-Werte ergänzt.

Einige der Events werden durch die IOLib automatisch erfasst. Weitere Details hierzu finden sich in Kapitel 4.3.1 (*Automatisch durch die SZM-Library gemessene Events*).

Parameter:

- **EventType (mandatory)**

Das zu erfassende Event. Die IOLib stellt Konstanten innerhalb des Datentyps „IOLEventType“ zur Verfügung:

  - IOLEventTypeView
  - IOLEventTypeData
  - IOLEventTypeDocument
  - IOLEventTypeDownload
  - IOLEventTypeGame
  - IOLEventTypeGesture

- IOLEventTypeHardwareButton
- IOLEventTypeIAP
- IOLEventTypeLogin
- IOLEventTypeAudio
- IOLEventTypeVideo
- IOLEventTypePush
- IOLEventTypeUpload
- IOLEventTypeWatchKit

- **EventState (mandatory)**

Die einzelnen Events können verschiedene Zustände einnehmen. So kann ein Download z.B. gestartet, durch den User abgebrochen, erfolgreich durchgeführt oder fehlerhaft beendet worden sein.

Folgende Zustände innerhalb einer Event-Kategorie können gemessen werden:

EVENT: View

- IOLViewAppeared
- IOLViewRefreshed
- IOLViewDisAppeared

EVENT: Data

- IOLDataCancelled
- IOLDataRefresh
- IOLDataSucceeded
- IOLDataFailed

EVENT: Document

- IOLDocumentOpen
- IOLDocumentEdit
- IOLDocumentClose

EVENT: Download

- IOLDownloadCancelled
- IOLDownloadStart
- IOLDownloadSucceeded
- IOLDownloadFailed

EVENT: Game

- IOLGameAction
- IOLGameStarted
- IOLGameFinished
- IOLGameWon
- IOLGameLost
- IOLGameNewHighscore
- IOLGameNewAchievement

- EVENT: Gesture
  - IOLGestureShake

EVENT: HardwareButton

- IOLHardwareButtonPushed

EVENT: IAP

- IOLIAPStarted
- IOLIAPFinished
- IOLIAPCancelled

EVENT: Login

- IOLLoginSucceeded
- IOLLoginFailed
- IOLLoginLogout

EVENT: Audio

- IOLAudioPlay
- IOLAudioPause
- IOLAudioStop
- IOLAudioNext
- IOLAudioPrevious
- IOLAudioReplay
- IOLAudioSeekBack
- IOLAudioSeekForward

EVENT: Video

- IOLVideoPlay
- IOLVideoPause
- IOLVideoStop
- IOLVideoNext
- IOLVideoPrevious
- IOLVideoReplay
- IOLVideoSeekBack
- IOLVideoSeekForward

EVENT: Push

- IOLPushReceived

EVENT: Upload

- IOLUploadCancelled
- IOLUploadStart
- IOLUploadSucceeded
- IOLUploadFailed

Weitere Details zu den messbaren Events und der dazugehörigen States sind in Kapitel 4.3 (*Events*) beschrieben.

- **Category (optional): Inhaltscode**

Der Inhaltscode wird im Parameter “category“ übermittelt. Dieser Code wird vom Anbieter selbst festgelegt. Die syntaktischen Vorgaben sind Kapitel 4.2 (*Richtlinien zur Vergabe der Codes*) zu entnehmen. Der Code dient zur inhaltlichen Kennzeichnung des angezeigten Content und wird vom Anbieter im INFOnline Kundencenter dem IVW Kategoriensystem 2.0 zugeordnet.

Der Anbieter entscheidet anhand der im Kapitel 4 (*Vorgaben zum Aufruf der SZM-Library*) beschriebenen Richtlinien, ob ein Event eine mobile PI im Sinne der IVW Richtlinien darstellt. Wenn ein Event unter die Definition einer mobilen PI fällt, ist zwingend ein Inhaltscode mitzugeben. Stellt ein Event keine mobile PI dar, soll **nil** übergeben werden. Die Länge dieses Feldes ist nicht beschränkt.

- **Kommentar**

Kommentarfeld. Die Länge dieses Feldes ist nicht beschränkt.
Übergabe dieses Wertes ist optional, ist er nicht definiert, soll **nil** übergeben werden.

Beispiele:

- **IOLEventTypeView / IOLViewAppeared**

Objective-C:

```
@implementation ViewController

- (void)viewDidAppear:(BOOL)animated {

    [super viewDidAppear:animated];

    [[IOLSession defaultSession] logEventWithType:IOLEventTypeView
                                 state:IOLViewAppeared
                                 category:@"Home"
                                 comment:nil];

    // Other Code ..
}
```

Swift:

```
class ViewController: UIViewController {

   @IBAction func playMusic(sender: AnyObject) {

      IOLSession.defaultSession().logEventWithType(IOLEventType.Audio,
         state: IOLAudioPlayed
         category: "Audio"
         comment: "Audio playback")

      // Other Code ..
   }
}
```

- **IOLEventTypeView / IOLViewRefreshed**

Objective-C:

```
@implementation ViewController

- (IBAction)refresh:(id)sender {

[[IOLSession defaultSession] logEventWithType:IOLEventTypeView
                             state:IOLViewRefreshed
                             category:@"Home"
                             comment: @"AdBanner shown"];

    // Other Code ..
}
```

Swift:

```
class ViewController: UIViewController {

   @IBAction func refresh(sender: AnyObject) {

      IOLSession.defaultSession().logEventWithType(IOLEventType.View,
         state: IOLViewRefreshed
         category: "Home"
         comment: "AdBanner shown")

      // Other Code ..
   }
}
```

- **IOLEventTypeAudio / IOLAudioPlay**

Objective-C:

```
@implementation ViewController

- (IBAction)playMusic:(id)sender {

[[IOLSession defaultSession] logEventWithType:IOLEventTypeAudio
                              state:IOLAudioPlay
                              category:@"Audio"
                              comment:@"Audio Playback"];

   // Other Code ..
}
```

Swift:

```
class ViewController: UIViewController {

   @IBAction func playMusic(sender: AnyObject) {

      IOLSession.defaultSession().logEventWithType(IOLEventType.Audio,
         state: IOLAudioPlayed
         category: "Audio"
         comment: "Audio playback")

      // Other Code ..
   }
}
```

3.4.3 Versand der Messdaten

- (void)sendLoggedEvents;

Die IOLib steuert den Versand der Messdaten selbständig und völlig transparent für den Enduser. Um den Versand der Daten zu forcieren, **kann sendLoggedEvents** aufgerufen werden. Die IOLib versucht dann, die Messdaten sofort bzw. alternativ nochmals zu versenden, sobald eine Datenverbindung aufgebaut wurde.

Parameter:

- -

Beispiel:

Objective-C:

```
@implementation ViewController

- (IBAction)send:(id)sender {
    [[IOLSession defaultSession] sendLoggedEvents];
}
```

Swift:

```
class ViewController: UIViewController {

    @IBAction func send(sender: AnyObject) {
        IOLSession.defaultSession().sendLoggedEvents()
    }
}
```

3.4.4 Session beenden

- (void)terminateSession;

Die aktive Session der IOLib kann explizit beendet werden. Dies ermöglicht ein Opt-out während der App-Laufzeit. Die bis dahin erfassten Daten werden nicht mehr versendet. **HINWEIS**: **Nur bei Opt-out durch den Nutzer verwenden!**

Beispiel:

Objective-C:

```
@implementation ViewController

- (void)disableIOLSession {
    [[IOLSession defaultSession] terminateSession];
}
```

Swift:

```
class ViewController: UIViewController {

    func disableIOLSession() {
        IOLSession.defaultSession().terminateSession()
    }
}
```

HINWEIS: Die IOLib Session muss anschließend neu gestartet werden! Das Vorgehen ist in Kapitel 3.2.1 (*Start einer Session*) beschrieben.

3.4.5 Einbindung Opt-out-Funktion

Den Nutzer einer App muss eine Opt-out Funktion gegeben werden. Die Implementierung obliegt dem Entwickler der jeweiligen App und sollte bei Aktivierung durch den Benutzer dazu führen, dass die SZM-Library entweder gar nicht initialisiert wird oder die laufende Session explizit beendet wird. Das Vorgehen ist in Kapitel 3.2.4. (*Session beenden*) beschrieben.

Nach Einbindung der Funktion können die Nutzer der App das Opt-out aktivieren und deaktivieren. Sofern Opt-out aktiviert wird, wird kein Zählimpuls ausgelöst.

HINWEIS: **Wird die laufende Session explizit beendet, dann werden alle bis dahin erfassten, aber noch nicht versandten Messdaten verworfen.**

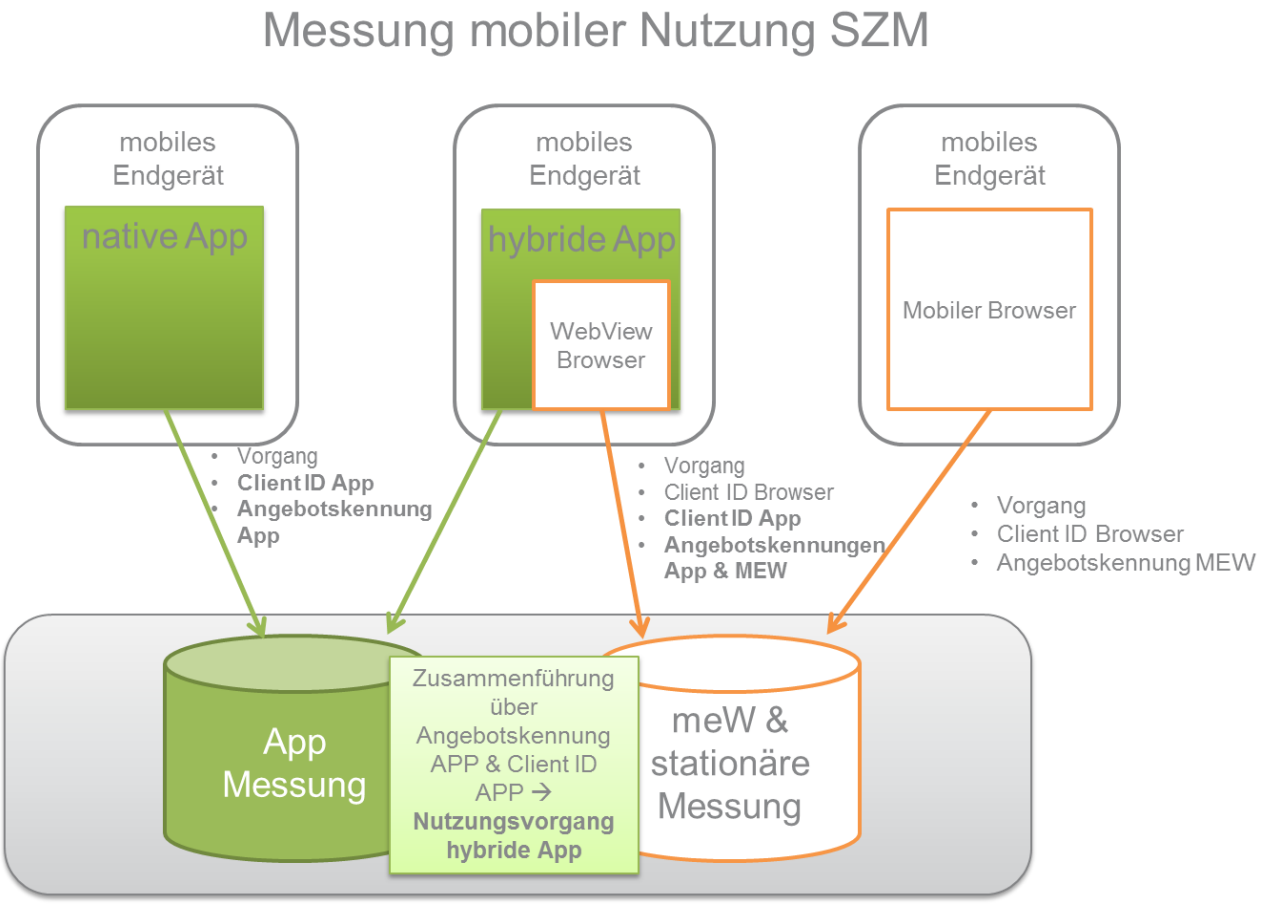
Wird das Opt-out revidiert, dann sollte die MessLib wieder gestartet werden. Das Vorgehen ist in Kapitel 3.2.1 (*Start einer Session*) beschrieben.

3.5 Hybrid-Messung

Die SZM-Library ist in der Lage, die Nutzung von Hybrid-Apps zu messen, d.h. auch Nutzeraktionen innerhalb mobiler Inhalte, welche in sog. WebViews dargestellt werden, können erfasst und mit den Messdaten des nativen App-Rahmens zusammengeführt werden.

Voraussetzung hierfür ist, dass die über den WebView aufgerufenen Webseiten ebenfalls mit dem SZM-Tag für mobile enabled Websites vertaggt sind. Außerdem muss der von der SZM-Library bereitgestellte WebView verwendet werden, um die beiden Messdatensätze aus der App-Messung und aus der MEW-Messung zusammenzuführen. Die App und die aus der App aufgerufenen MEWs müssen unterschiedliche, von INFOnline vergebene Angebotskennungen verwenden.

Nachfolgendes Schaubild zeigt eine Übersicht mobiler Nutzung im SZMnG:



HINWEIS Hybride Apps beziehen den externen Web-Content i.d.R. von Websites, die für die Nutzung von mobilen Endgeräten optimiert sind. Diese Websites werden in diesem Dokument als MEW (=mobile enabled Website) bezeichnet. Sollte Ihre hybride App Content aus einer stationären Website (optimiert für die Nutzung von PCs und Notebooks) beziehen, gelten alle Bedingungen, die im Dokument *INFOnline Integration Guide SZM-Tag* beschrieben sind.

Um die Messung von Hybrid-Apps zu ermöglichen, müssen Webseiten in der App durch einen speziellen WebView, den **IOLWebView**, aufgerufen werden. Hierzu muss bei jeder Initialisierung eines WebViews im Source Code ein **IOLWebView** anstelle eines Standard **UIWebView** erzeugt werden.

Anschließend kann der View wie ein regulärer **UIWebView** behandelt werden. Ein **IOLWebView** leitet direkt vom **UIWebView** ab, bietet nach außen keinerlei neue APIs und verhält sich somit vollkommen transparent.

Wird der **UIWebView** im InterfaceBuilder erzeugt, muss lediglich der Klassenname des Objekts von **UIWebView** auf **IOLWebView** geändert werden.

HINWEIS: **Bei einem IOLWebView ist es nicht möglich, die delegate Property auf Pointer-Gleichheit zu überprüfen. Ebenso ist die Delegate Property eines IOLWebView niemals nil. Der setter kann allerdings wie gewohnt benutzt werden.**

Beispiel:

Objective-C:

```
UIWebView* webView = [[IOLWebView alloc] init];

webView.delegate = self; // this works and the delegate is called like in a regular
                         // UIWebView

if (webView.delegate == self) {
   NSLog(@"I am never called!");
}

webView.delegate = nil; //your delegate doesn't receive any calls from this point on

if (webView.delegate == nil) {
   NSLog(@"I am never called, either!");
}
```

Swift:

```
let webView: UIWebView = IOLWebView()

        webView.delegate = self // this works and the delegate is called like in a
regular UIWebView

        if webView.delegate === self {
            NSLog("I am never called!")
        }

        webView.delegate = nil //your delegate doesn't receive any calls from this
point on

        if webView.delegate == nil {
            NSLog("I am never called, either!")
        }
```

HINWEIS: Dieses Beispiel produziert keinen Log auf der Konsole! Statt dessen sollte mittels `isEqual:` oder `isKindOfClass:` geprüft werden, ob es sich um das erwartete delegate Objekt handelt.

3.5.1 Unterstützung WKWebView (WebKit)

Mit Version 1.1.9 der INFOnlineLibrary wurde eine neue Klasse **IOL_WKWebView** ergänzt. Diese Klasse erweitert die Klasse **WKWebView** um die gleiche Funktionalität, um die die Klasse **IOL_WebView** die Klasse **UIWebView** erweitert. Entsprechend gelten bei der Verwendung von **IOL_WKWebView** die gleichen Besonderheiten (z.B. Beachtung der Pointer-Gleichheit) wie bei **IOLWebView**.

Die bisherige Klasse **IOLWebView** wurde für eine klarere Unterscheidung in **IOL_UIWebView** umbenannt. Damit an existierenden Verwendungen von **IOLWebView** keine Änderungen erforderlich sind, steht **IOLWebView** als Subklasse von **IOL_UIWebView** weiterhin zur Verfügung.

Im Vergleich zur bisherigen Implementierung von **IOLWebView** ist u.A. die Methode **isWK-WebViewAvailable** hinzu gekommen:

```
#import "IOL_UIWebView.h"

@interface IOLWebView : IOL_UIWebView

+ (BOOL)isWKWebViewAvailable;

@end
```

Da WebKit erst in iOS 8.0 eingeführt wurde, kann mit dieser Methode zur Laufzeit geprüft werden, ob das WebKit zur Verfügung steht und ob eine Instanz von **IOL_WKWebView** statt **IOLWebView** bzw. **IOL_UIWebView** erstellt werden. Im Zweifelsfall kann immer **IOLWebView** verwendet werden.

Um den Wechsel bzw. Austausch von **IOL_WKWebView** und **IOL_UIWebView** möglichst einfach zu gestalten, wurde ein neues Protokoll **IOLWebViewCommon** erstellt, welches einige gemeinsame Properties und Methoden beider WebViews definiert. Damit ist es beispielsweise möglich, ein generisches **UIView** mit Konformität zu **IOLWebViewCommon** zu deklarieren und zur Laufzeit dynamisch mit **IOL_WKWebView** und **IOL_UIWebView** zu initialisieren, ohne dass Compiler-Fehler oder –Warnungen auftreten. Dies ist z.B. sinnvoll, wenn das Programm auf Geräten mit iOS Version 8.0 (oder neuer) den neueren **IOL_WKWebView** verwenden, aber auch unter älteren iOS Versionen funktionieren und entsprechend den **IOL_UIWebView** als Fallback verwenden soll.

Nachfolgend ist ein Beispiel aufgeführt, das die Verwendung des Protokolls **IOLWebViewCommon** in Verbindung mit dem dynamischen Wechsel zwischen **IOL_WKWebView** und **IOL_UIWebView** veranschaulicht:

Objective-C:

```
#import "BrowserViewController.h"

@interface BrowserViewController ()

@property UIView<IOLWebViewCommon> *webView;

@end

@implementation BrowserViewController

- (void)viewDidLoad {

    [super viewDidLoad];

    if ([IOLWebView isWKWebViewAvailable]) {
        self.webView = [[IOL_WKWebView alloc] initWithFrame:self.view.frame];
    } else {
        self.webView = [[IOL_UIWebView alloc] initWithFrame:self.view.frame];
    }

    [self.view addSubview:self.webView];

    [self.webView setDelegate:self];

    [self.webView loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString:@"http://www.google.com"]]];
}
```

Swift:

```swift
import UIKit

class BrowserViewController: UIViewController {

    var webView: IOLWebViewCommon = IOLWebView()

    override func viewDidLoad() {

        super.viewDidLoad()

        if IOLWebView.isWKWebViewAvailable() {
            webView = IOL_WKWebView(frame: self.view.frame)
            self.view.addSubview(webView as! IOL_WKWebView)
        } else {
            webView = IOL_UIWebView(frame: self.view.frame)
            self.view.addSubview(webView as! IOL_UIWebView)
        }

        self.webView.setDelegate(self)

        self.webView.loadRequest(NSURLRequest(URL: NSURL(string: "http://www.google.com")!))
    }
}
```

3.6 Debug-Informationen

Zum Zweck der allgemeinen Fehleranalyse und insb. des Versands der Messdaten kann die SZM-Library in einen Debug-Modus versetzt werden. In diesem Debug-Modus erzeugt die SZM-Library Ausgaben im Logstrom (Konsole).

Objective-C:

```objc
- (void)applicationDidFinishLaunching:(UIApplication*)application {

    [[IOLSession defaultSession] setDebugLogLevel:IOLDebugLevelInfo];
```

Swift:

```swift
override func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject : AnyObject]?) -> Bool {

    IOLSession.defaultSession().setDebugLogLevel(3) //IOLDebugLevelInfo
```

Art und Umfang der Logausgaben können über einen DebugLevel bestimmt werden.

Folgende Debug-Level sind definiert:

- **IOLDebugLevelOff**
  - Logausgabe: **Deaktiviert (Default)**
- **IOLDebugLevelError**
  - Logausgabe: nur Fehler
- **IOLDebugLevelWarning**
  - Logausgabe: Fehler und Warnungen

- **IOLDebugLevelInfo**
  - Logausgabe: Fehler, Warnungen und Infos
- **IOLDebugLevelTrace**
  - Logausgabe: Fehler, Warnungen, Infos, Events, Requests und Responses

HINWEIS Requests und Responses werden im Documents Folder gespeichert. Zu Debugging-Zwecken kann in iTunes Filesharing eingeschaltet werden und die entsprechenden Dateien einfach über iTunes kopiert werden.

4 Vorgaben zum Aufruf der SZM-Library

4.1 Allgemeines

Die Erfassung der App-Nutzung durch den Benutzer erfolgt, indem die App die SZM-Library bei definierten Ereignissen, welche eine Nutzer-Interaktion kennzeichnen, aufruft.
Die Nutzer-Interaktion wird als Event bezeichnet.

HINWEIS: **Die SZM-Library muss von der APP bei Eintreten des Events explizit aufgerufen werden.**

Weiterhin misst die IOLib bestimmte System- oder App-spezifische Werte automatisch. Zu diesem Zweck muss die Integration der IOLib iOS exakt wie in Kapitel 3.1.3. (*Integration des IOLib iOS Frameworks*) beschrieben erfolgen.

4.1.1 Interpretation von Events als mobile PI

Die nachfolgend aufgeführten Vorgaben definieren, wie die SZM-Library im Kontext der SZM Mobile Applications Messung zu verwenden ist.

Aus technischer Sicht wird zwischen 2 Typen von Events unterschieden:

1. **PI-Events**

Bei den PI-Events wird der Event dazu benutzt, analog zum stationären Web, eine PageImpression zu erzeugen. Diesem Event muss ein Inhaltscode (in der Folge einfach als „Code“ bezeichnet) zugeordnet werden. Dieser Code kann anschließend den unterschiedlichen Kategorien zugeordnet werden und dient als Grundlage für die Bildung von Belegungseinheiten.
Bei den PI-Events sind die Vorgaben zur Mobile Impression der IVW zu beachten:

„Eine Mobile Impression ist eine Nutzeraktion innerhalb eines mobilen Angebots, die zu einem Aufruf eines Werbemittels führt oder führen könnte. Jede Nutzeraktion darf nur einmal gezählt werden. Nutzeraktionen, die zu keiner potentiellen Werbeauslieferung führen, dürfen nicht gezählt werden.

Voraussetzungen für die Zuweisung einer MI zu einem Angebot:

Der ausgelieferte Inhalt muss (bei mobile enabled Websites) den FQDN bzw. (bei Apps) den App-Namen des Angebots (oder Alias/Redirect) oder den zugewiesenen MEW- oder App-Namen des Angebots tragen.

Nutzeraktion:

Eine MI wird ausgelöst durch eine vom Nutzer durchgeführte Aktion. Darunter fallen ebenfalls: Reload, Öffnen einer App, Öffnen eines Browsers.

Keine Nutzeraktion:

Aufruf eines Inhalts durch eine automatische Weiterleitung (außer Redirects und Alias)., automatischer Reload, das Aufrufen eines Inhaltes beim Schließen (auch: Background) eines Browserfensters oder einer App, das Aufrufen von Inhalten über Robots/Spider und Ähnliches.

Keine Mobile Impression:

Das Scrollen innerhalb eines bereits geladenen Inhalts.

2. Non-PI- Events

Non-PI-Events sind Nutzeraktionen, die als Event im SZM-System erfasst werden, jedoch nicht zur Zählung einer Mobile Impression führen. Diesem Event darf kein Code zugeordnet werden. Beispiele für Non-PI-Events sind

- automatisch von der IOLib erfasste Events
- vom Anbieter festgelegte Events, welche keine mobile PI darstellen und dennoch als Events gemessen werden sollen, um z.B. die Nutzung der App durch die Nutzer besser nachvoll-ziehen zu können.

4.2 Richtlinien zur Vergabe der Codes

Bei den PI-Events ist der Code als eindeutige Kennzeichnung des angezeigten Inhalts mitzugeben. Der Code wird vom App-Anbieter spezifiziert.

Bei der Spezifikation der Inhalts-Codes sind die Code-Richtlinien der INFOnline zu beachten

- Länge der Codes: Ein Code darf maximal 255 Zeichen enthalten
- Anzahl der Codes: es dürfen maximal 2000 Codes verwendet werden
- Erlaubte zeichen: a-z, A-Z, 0-9; Komma „,“; Bindestrich „-“; Unterstrich „_“; Slash „/“

Eine ausführliche Dokumentation der INFOnline Code-Richtlinien finden Sie im „INFOnline Configuration Guide“ unter:

https://www.infonline.de/downloads

4.3 Events

In den nachfolgenden Tabellen sind Events aufgeführt, welche in der Messung erhoben werden bzw. erhoben werden können. Unter welchen Umständen ein Event zu einer PageImpression führen kann, wird im folgenden ebenfalls erläutert.

4.3.1 Automatisch durch die SZM-Library gemessene Events

Die nachfolgende Tabelle beschreibt die Events, bei denen die SZM-Library automatisch

aufgerufen wird. Bei den Events handelt es sich um Nutzeraktionen, welche aus technischen Gründen erhoben werden, jedoch nicht zur Zählung einer Mobile Impression führen. Diesem Event darf kein Code zugeordnet werden.

Event Type	Event State	Event	Bemerkung
IOLEventTypeApplication	IOLApplicationStart, IOLApplicationEnterBackground, IOLApplicationEnterForeground, IOLApplicationResignActive, IOLApplicationBecomeActive, IOLApplicationTerminate, IOLApplicationCrashed	App-spezifische Event, z.B. Start der App, Beenden der App, Crash, etc.	ResignActive: Eingehender Anruf, Push-Notification Alert, Timer Alarm, etc. EnterFore/Background: App geht in den Hintergrund Terminate: App wird beendet
IOLEventTypeBackgroundTask	IOLBackgroundTaskStart IOLBackgroundTaskEnd	Ein Hintergrundprozess wird gestartet bzw. beendet	Download oder Upload größerer Dateien, die evtl. im Hintergrund weiterlaufen sollen
IOLEventTypeOpenApp	IOLOpenAppMaps IOLOpenAppOther	Eine andere App wird gestartet bzw. App wird über eine URL verlassen	Maps: Google Maps (iOS4/5) bzw Maps (>=iOS6) wird aufgerufen Other: Andere Apps bzw. URIs werden aufgerufen (Email, Phone, Websites, etc.)
IOLEventTypeInternetConnection	IOLInternetConnectionEstablished IOLInternetConnectionLost IOLInternetConnectionSwitchedInterface	Art der Konnektivität ändert sich	Verbindung aufgebaut bzw. verloren Wechsel von Mobile auf Wifi bzw. umgekehrt

IOLEventTypeWebView	IOLWebViewInit	hybrid-Messung wird aktiviert	

Die „automatisch durch die Lib gemessenen Events“ werden erfasst, sobald die Klasse „IOLApplication“ der MessLib als Basis der App verwendet wird.

4.3.2 PI-Events

Im Folgenden sind Events aufgeführt, welche typischerweise zur Auslösung einer PI führen. Die Übernahme des Schemas wird empfohlen. Die Events müssen manuell ausgelöst werden. Das Vorgehen ist in Kapitel 3.2.2 (*Logging eines Events*) beschrieben. Eine automatische Erhebung erfolgt nicht. PI-Events muss ein Code zugeordnet werden. Dieser Code kann anschließend den unterschiedlichen Kategorien zugeordnet werden und dient als Grundlage für die Bildung von Belegungseinheiten. Bei den PI-Events sind die Vorgaben zur Mobile Impression der IVW zu beachten.

Event Type	Event State	Event	Bemerkung
IOLEventTypeDeviceOrientation	IOLEventTypeDeviceOrientation	Ausrichtung des Gerätes	LandscapeLeft / LandscapeRight oder Portrait / Portrait UpsideDown
IOLEventTypeGesture	IOLGestureShake	Gerät wird geschüttelt	
IOLEventTypeView	IOLViewAppeared IOLViewRefreshed	Ein View (aka „Page“) wurde angezeigt oder mit neuen Daten aktualisiert	Beispiele: Appeared: initialer Aufruf einer Seite Refreshed: Suchfilter o. Aktualisierung von Daten
IOLEventTypeGame	IOLGameAction IOLGameStarted	Gaming-Events	Aktion innerhalb eines Spiels Spiel gestartet
IOLEventTypeAudio	IOLAudioPlay IOLAudioPause IOLAudioStop IOLAudioNext IOLAudioPrevious IOLAudioReplay IOLAudioSeekBack IOLAudioSeekForward	Audio Playback	Wiedergabe, Pause, Stop, Nächster/vorheriger Titel, Vor/Zurückspulen, Wiederholung (Übergabe des Parameters Audio oder Video beim EventLogging verpflichtend)

IOLEventTypeVideo	IOLVideoPlay IOLVideoPause IOLVideoStop IOLVideoNext IOLVideoPrevious IOLVideoReplay IOLVideoSeekBack IOLVideoSeekForward	Video Playback	Wiedergabe, Pause, Stop, Nächster/vorheriger Titel, Vor/Zurückspulen, Wiederholung (Übergabe des Parameters Audio oder Video beim EventLogging verpflichtend)

4.3.3 Non-PI-Events

Im Folgenden sind Events aufgeführt, welche typischerweise nicht zur Zählung einer PI führen. Sollten jedoch im Einzelfall Umstände vorliegen, unter denen auch hier eine Mobile PI erzeugt werden soll, können diese Events benutzt werden, um dies zu ermöglichen. Bevor Sie diese Events zur Erzeugung von PIs benutzen, klären Sie den jeweiligen Sachverhalt bitte unmittelbar mit der IVW Geschäftsstelle ab. Sollen diese Events zur Zählung von PIs führen, muss das auch hier manuell ausgelöst werden. Diesem Event muss dann ein Code zugeordnet werden. Dieser Code kann anschließend den unterschiedlichen Kategorien zugeordnet werden und dient als Grundlage für die Bildung von Belegungseinheiten.

Event Type	Event State	Event	Bemerkung
IOLEventTypeView	IOLViewDisAppeared	Ein View (aka „Page“) wurde verlassen	Beispiele: DisAppeared: Screen verlassen
IOLEventTypeDocument	IOLDocumentOpen IOLDocumentEdit IOLDocumentClose	Dokument / Liste Bearbeitung	Liste editiert Dokument gespeichert
IOLEventTypeData	IOLDataCancelled IOLDataRefresh IOLDataSucceeded IOLDataFailed	Datenverbindung/ -verarbeitung	Datenverbindung abgebrochen Daten wurden aktualisiert Daten wurden erfolgreich übertragen Daten wurden nicht übertragen
IOLEventTypeDownload	IOLDownloadCancelled IOLDownloadStart IOLDownloadSucceeded IOLDownloadFailed	Download von Daten	Download wurde initiiert Download wurde abgebrochen Download erfolgreich beendet Download fehlgeschlagen
IOLEventTypeUpload	IOLUploadCancelled IOLUploadStart IOLUploadSucceeded IOLUploadFailed	Upload von Daten	Upload wurde initiiert Upload wurde abgebrochen Upload erfolgreich beendet Upload fehlgeschlagen

IOLEventTypeLogin	IOLLoginSucceeded IOLLoginFailed IOLLoginLogout	Login	Login erfolgreich durchgeführt Login fehlgeschlagen Logout durchgeführt/Session beendet
IOLEventTypeHardwareButton	IOLHardwareButtonPushed	Schalter oder Knopf am Gerät gedrückt	Lautstärke über Schalterwippe am Gerät geändert Device gelocked (Power Taste betätigt) Druck auf Home-Button

Event Type	Event State	Event	Bemerkung
IOLEventTypeGame	IOLGameFinished IOLGameWon IOLGameLost IOLGameNewHighscore IOLGameNewAchievement	Gaming-Events	Spiel beendet Spiel(runde) gewonnen Spiel(runde) verloren Neuer Highscore erreicht Neues Achievement erreicht

Sobald in der App ein unter „manuell auszulösende Events“ beschriebenes Ereignis ausgelöst wird, ist die SZM-Library per **logEventWithType** aufzurufen. Hierbei ist die in der Spalte „Event Type“ sowie „Event State“ gelistete Konstante als Parameter zu übergeben (siehe dazu auch Kapitel 3.2 (*SZM-Library Funktionen*)).

Die Übermittlung der Non-PI-Events an das SZM System ist optional. Die Non-PI-Events haben keinen Einfluss auf die Reichweitenermittlung Ihrer App.

Diese Events können von Ihnen zur quantitativen Ermittlung der Häufigkeit des Auftretens dieser Events (in den INFOnline Auswertesystemen) verwendet werden. Hierbei ist zu beachten, dass die Erfassung und Übermittlung der Non-PI-Events technische Ressourcen (CPU-Zeit, Netzwerkverkehr, Batterie) auf dem Endgerät verwendet.

Mit Hinblick auf die Inanspruchnahme der technischen Ressourcen auf dem Endgerät bitten wir Sie, in der Planung der Umsetzung der Integration der MessLibs zu entscheiden, ob Ihre App NonPI-Events an das SZM-System übermitteln soll.

4.3.4 Sonderfall Event „ViewRefreshed“

Für den Fall, dass eine View refreshed wird (Event IOLEventTypeView, State IOLViewRefreshed), ist Folgendes zu beachten:

HINWEIS Das Event darf nur geloggt werden (bzw. die SZM-Library aufgerufen werden), wenn der Refresh der Daten manuell durch den Nutzer ausgelöst wurde. Bei einem automatischen Refresh darf das Event nicht geloggt werden.

4.3.5 Sonderfall WatchKit Events

Für die Messung von WatchKit Events in der WatchKit Extension können Events nicht direkt über eine Session geloggt werden. Hierfür muss die Klasse **IOLWatchKitHelper** verwendet werden, welche die Kommunikation der WatchKit Extension mit der Parent Application (App auf dem iOS Gerät) übernimmt und die Events an diese weitergibt.

Das Pendant zur Klasse **UIViewController** ist die Klasse **WKInterfaceController** im WatchKit. Diese Klasse enthält die beiden Methoden **willActivate** und **didDeactivate**, welche die Pendants zu den Methoden **viewWillAppear** bzw. **viewDidDisappear** aus **UIViewController** darstellen.

Um die serverseitige Erfassung von WatchKit Events jedoch so einfach wie möglich zu gestalten, sollte der Eventtyp **IOLEventTypeView** und dessen Events **IOLViewAppeared** und **IOLViewDisappeared** in den Callback-Methoden der Klasse **WKInterfaceController** verwendet werden.

Beispiel (Code in einer Subklasse von **WKInterfaceController**):

Objective-C:

```
- (void)willActivate {
    // This method is called when watch view controller is about to be visible to user
    [super willActivate];

    IOLEvent *event = [[IOLEvent alloc] initWithType:IOLEventTypeView
                                               state:IOLViewAppeared
                                            category:NSStringFromClass([self class])
                                             comment:nil];

    [IOLWatchKitHelper logEvent:event];
}

- (void)didDeactivate {
    // This method is called when watch view controller is no longer visible
    [super didDeactivate];

    IOLEvent *event = [[IOLEvent alloc] initWithType:IOLEventTypeView
                                               state:IOLViewDisappeared
                                            category:NSStringFromClass([self class])
                                             comment:nil];

    [IOLWatchKitHelper logEvent:event];
}
```

Swift:

```
5       override func willActivate() {
6           // This method is called when watch view controller is about to be visible to user
7           super.willActivate()
8
9           let event: IOLEvent = IOLEvent(type: IOLEventType.View,
10              state: IOLViewAppeared,
11              category: NSStringFromClass(self.dynamicType),
12              comment: nil)
13
14          IOLWatchKitHelper.logEvent(event)
15      }
16
17      override func didDeactivate() {
18          // This method is called when watch view controller is no longer visible
19          super.didDeactivate()
20
21          let event: IOLEvent = IOLEvent(type: IOLEventType.View,
22              state: IOLViewDisappeared,
23              category: NSStringFromClass(self.dynamicType),
24              comment: nil)
25
26          IOLWatchKitHelper.logEvent(event)
27      }
```

5 Kontakt

Das Service & Support-Team ist werktags von 9 bis 18 Uhr erreichbar via

Telefon:	0228 / 410 29 – 77
E-Mail für organisatorische Anfragen:	service@INFOnline.de
E-Mail für technische Anfragen:	support@INFOnline.de