IBM Tealeaf CX Mobile
Version 9 Release 0.1
December 4, 2014

IBM Tealeaf CX Mobile iOS Logging Framework Guide

Note

Before using this information and the product it supports, read the information in "Notices" on page 99.

This edition applies to version 9, release 0, modification 1 of IBM Tealeaf CX Mobile iOS Logging Framework and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

IBM Tealeaf CX Mobile iOS Logging Framework

The IBM Tealeaf CX Mobile iOS Logging Framework for mobile native applications requires the IBM Tealeaf CX Mobile license for Mobile App.

For more information, contact your IBM Tealeaf representative. Licensees must implement in their apps code that is provided by IBM Tealeaf. For more information on downloading IBM Tealeaf, see IBM® Passport Advantage® Online.

The *IBM Tealeaf CX Mobile iOS Logging Framework Guide* provides guidance on how to enable the capture of mobile application data directly from the application that is installed on the visitor's iOS-enabled device.

Note: Whenever possible, use the latest version of the IBM Tealeaf CX Mobile iOS Logging Framework software.

Chapter 1. Introduction

Introduction

With the IBM Tealeaf CX Mobile iOS Logging Framework, you instrument your native and hybrid iOS applications for logging and analysis. It captures device context and user activity, so you can monitor and evaluate the performance of your applications.

It was designed for simple implementation: it uses standard iOS classes and user interface controls to track user interface events, and minimizes the impact on your application's performance. Even without the framework, Tealeaf® can monitor the traffic between your application and your server. With the framework, you get unprecedented insight into the performance of your application.

Framework version

Before you contact IBM technical support, retrieve the version of the IBM Tealeaf CX Mobile iOS Logging Framework that you are using.

Server-side

If your installation of IBM Tealeaf CX Mobile iOS Logging Framework is submitting posts to the server, you can retrieve the version from the `[env]` section of the request:

```
X-Tealeaf: device (iOS) Lib/8.5.6.1
```

In this example, the version number is `8.5.6.1`.

Runtime

Developers can retrieve the version number by using the `frameworkVersion` method.

Features

Automatic logging

After you link in the framework and change 2 lines of your application's code, the framework can record device context and user actions like button taps and navigation.

Table 1. Automatic logging

Context	Notifications	Control Events
• Network status • Device type • Operating system version	• App lifecycle • Table view row selection • Text view and text field changes	• View controller loading, appearing • Button taps • Alert view and action sheets • HTTP and web view activity

Customized data analysis

You can create your own custom log events to mark activity to analyze.

If needed, you can mark the beginning of sessions for analysis and share session identifiers between the framework and your application's own network traffic.

Kill switch

To manage traffic volume, you can enable the kill switch. You can set up the CX Mobile iOS Logging Framework to check your server when the application starts and enable or disable the kill switch.

Privacy and security

Sensitive user input can be omitted or masked, and you can disable local storage completely.

HTTPS is supported for transmitting data.

How it works

The framework is configurable, efficient, and secure.

Data capture

To detect data for different types of events, IBM Tealeaf CX Mobile iOS Logging Framework uses different methods.

- The framework listens for global notifications from iOS.
- The framework logs button events through `sendEvent:` and `sendAction:to:from:forEvent:` methods in default `TLFApplication class(subclassing UIApplicaion)` or your own customized `UIApplication` class.
- When no notification, event, or action is available, the framework accesses the Objective C run time so that standard iOS SDK classes can report user actions.

Data storage and communication

CX Mobile iOS Logging Framework packages data for periodic submission.

- Data is packaged to be sent in bursts, instead of at each event, according to sizes and times that you can configure.
- Data is sent by HTTPS or HTTP.
- Each submitted JSON message contains data from a single session only.
- Data is sent in JSON format.
- Data can be sent when the screen changes, at application startup, on going to the background, or when your application tells the framework to send data.
- The maximum cache size is configurable, including the option to avoid local storage completely.

Performance optimization

To optimize performance, CX Mobile iOS Logging Framework offers these options.

- Separate threads handle read and write operations to local storage, server interaction, collecting context data and formatting log entries.
- Initialization can be delayed after application launch.

Related documentation

As you implement and use the IBM Tealeaf CX Mobile iOS Logging Framework, you can reference any of the following documents.

Table 2. CX Mobile iOS Logging Framework documentation resources

Document	Description
IBM Tealeaf Client Framework Data Integration Guide	This document can be referenced by your IBM Tealeaf administrators whenever you are implementing an IBM Tealeaf client framework. **Note:** After you deploy your client framework, extra configuration can be necessary to capture the data in IBM Tealeaf, and to make the data available for creating events, which enables search and reporting.
IBM Tealeaf CX Mobile Administration Manual	Information for IBM Tealeaf administrators on the IBM Tealeaf CX Mobile product. Requires the IBM Tealeaf CX Mobile license.
IBM Tealeaf CX Mobile User Manual	User documentation for IBM Tealeaf CX Mobile. • "Search and Replay for Mobile App" • "Reporting for Mobile App" Requires the IBM Tealeaf CX Mobile license.
IBM Tealeaf CX Mobile Android Logging Framework Reference Guide	Installation and implementation guide for the IBM Tealeaf CX Mobile Android Logging Framework for Android-based mobile native applications. Requires the IBM Tealeaf CX Mobile license.
IBM Tealeaf CX UI Capture for AJAX	Installation and implementation guide for the IBM Tealeaf CX UI Capture for AJAX solution for AJAX-based web and mobile web applications. • Requires the IBM Tealeaf CX license. • Use of all mobile web functions requires IBM Tealeaf CX Mobile license.

Terminology

A glossary is available for terminology that is used in this guide and applicable to IBM Tealeaf.

For more information, see the *IBM Tealeaf Glossary*.

Next steps

To learn about the IBM Tealeaf CX Mobile iOS Logging Framework, read through the chapters in this guide and review the sample code.

You can also review "Tealeaf Configuration for Client Frameworks" in the *IBM Tealeaf Client Framework Data Integration Guide*.

Chapter 2. Tealeaf iOS Logging Framework Installation and Implementation

Use of the Tealeaf Logging Frameworks for mobile native applications requires the Tealeaf CX Mobile license for Mobile App. For more information, contact your Tealeaf representative. Licensees must implement in their apps code that is provided by Tealeaf. For more information on downloading IBM Tealeaf, see IBM Passport Advantage Online.

Client framework versions supported in this documentation

The installation and implementation instructions in this guide apply to the step-based version of JSON messaging from this client framework.

The installation and implementation instructions for the legacy version are similar, but require configuration in the Windows pipeline.

Integrate the IBM Tealeaf Mobile SDK with your iOS application

You integrate the IBM Tealeaf Mobile SDK with your iOS application so that the CX Mobile iOS Logging Framework can capture user interface and application events from your application. You integrate the SDK with every app from which you want to capture user interface and application events.

Installation package

IBM Tealeaf CX Mobile iOS Logging Framework is delivered in the IBM Tealeaf CX Mobile iOS Logging Framework 9.0 - iOS Logging Framework for Windows within the IBM Passport Advantage Online.

The package contains the following software components.

- `Tealeaf/Resources/TLFResources.bundle`. The Bundle file. This bundle contains all configuration files needed.
- `Tealeaf/Library/libTLFLib.a`. Library that is designed for use on iOS devices. Use this library for development and testing directly on the iOS device, and include it with your shipping application.
- `Tealeaf/Include` folder. This folder contains the header files that are required for customization purposes of the framework implementation.

Hardware and software requirements

To develop iOS applications effectively with CX Mobile iOS Logging Framework, the following hardware and software is required.

Consult Apple's iOS Dev Center for the most recent iOS technical documentation and tools.

- Intel based Mac for application development
- Mac OS X 10.6.6 or later
- Xcode 4 or later

Note: Apple no longer supports armv6 devices. The framework is compatible with armv7 or later devices.

- iOS SDK 5 for devices that run iOS 5.1.1 or later

  Note: If you use iOS SDK 7, there are limitations. Alert view button click events are not recorded or replayed. the Tab Bar has the style that is used in iOS SKD 6 when you replay iOS native mobile app sessions. Due to changes in the iOS 7 platform `[TLFCustomEvent logPrintScreenEvent]` no longer captures alert view dialogs. To capture the alert dialogs, you must run your own screen capture routine, then call `[TLFCustomEvent logImage:]` or `[TLFCustomEvent logImageSyncronous:]`.
- iTunes 10 or later

IBM Tealeaf client frameworks do not support forwarding of application data to third-party systems. Application data must be forwarded to the server that hosts the native application.

Impact on device resources

In benchmark tests, the CX Mobile iOS Logging Framework has the following effects on resources of the visitor's device.

- 2-3% more memory consumption
- Minimal effect on battery life

  Note: According to Apple, the API used to retrieve the battery level from the device can be out of sync with the value that displays on the device. See http://iphonedevelopertips.com/device/display-battery-state-and-level-of-charge.html. The value is also updated in 5% increments only. See http://www.iphonedevsdk.com/forum/iphone-sdk-development/14301-battery-level.html

Log screen layout for iOS mobile app session replay

You can replay a mobile app session in cxImpact Browser Based Replay as you would an HTML web session instead of viewing the mobile app session as a series of screen captures.

The screen layouts of the native mobile app sessions are captured in IBM Tealeaf JSON format. The screen layouts are then sent back to replay server. The replay server uses a template engine, which interprets the JSON into HTML format. You can then replay the screen lay out from the native mobile app session as HTML pages in cxImpact Browser Based Replay.

There are several advantages to using JSON data to replay mobile app session over screen captures.

- Reduce bandwidth. Screen captures for each screenview generate relatively large image data. It not only consumes large amounts of wireless and cellular bandwidth, but it also consumes more memory inside the device. It also impacts the app performance.
- Mask sensitive information. You cannot mask sensitive information in a screen capture. When using JSON data to replay mobile app sessions, you can mask `EditTexts` by adding `View IDs` to the `MaskIdList` attribute in `TLFConfigurableItems.properties`.
- Draw user interactions (UI events) onto the HTML pages that are created from the JSON data.

For more information on mobile ap session replay templates, see "Native app session replay customization" in the *IBM Tealeaf CX Configuration Manual.*

Replay logging can be automatic, manual, or a combination of the two. To enable automatic layout logging find `LogViewLayoutOnScreenTransition` in TLFConfigurableItems and set it to `YES`. This will automatically log a view controller when the view controller's `viewDidAppear:(BOOL)animated` method is called.

Note: If the viewController overrode the viewDidAppear, method[super viewDidAppear:animated] must be called.

Correct

```
-(void)viewDidAppear:(BOOL)animated
{
 [super viewDidAppear:animated];
 // Custom code
}
```

Incorrect

```
-(void)viewDidAppear:(BOOL)animated
{
 // Custom code
}
```

Several methods are included for manual logging of screen layout.

The following is the most basic manual logging method. The following method logs the layout of the viewController passed into it.

```
-(BOOL)logScreenLayoutWithViewController:(UIViewController *)viewController
```

The following method performs the same action as the first method, but you can pass in a specific name for the screen layout that is logged. This is helpful when you log a view controller that can perform several different functions.

```
-(BOOL)logScreenLayoutWithViewController:(UIViewController *)viewController
andName:(NSString *)name
```

The following method performs the same action as the first method, but after the specified delay. This is helpful for logging after certain events, such as reloading the data in a table. The delay is measured in seconds.

```
-(BOOL)logScreenLayoutWithViewController:(UIViewController *)viewController
       andDelay:(CGFloat)delay
```

The following method performs the same function as the previous method, but it allows you to pass in a name for the layout.

```
-(BOOL)logScreenLayoutWithViewController:(UIViewController *)viewController
       andDelay:(CGFloat)delay andName:(NSString *)name
```

In addition to logging the main view controller passed in, this method allows you to pass in an array of other views to be logged at the same time. This is useful in instances where there are elements on screen that are not part of the same view hierarchy, such as an overlay attached directly to the application's window or an alert view.

```
-(BOOL)logScreenLayoutWithViewController:(UIViewController *)viewController
andRelatedViews:(NSArray*)views
```

The following method performs the same action as the previous method, but it allows you to pass in a name for the layout.

```
-(BOOL)logScreenLayoutWithViewController:(UIViewController *)viewController
andRelatedViews:(NSArray*)views andName:(NSString *)name
```

Where and when to call manual logging

With automatic logging enabled, view controllers are logged during the viewDidAppear stage of the view lifecycle. If the view that is logged is loading remote data, this is not adequate. In this case, the ideal time to call the logging method is when the remote data is done loading and displaying.

```
- (void)RESTRequestCompleted:(RESTRequest *)request responseData:
(NSDictionary *)responseData response:(NSHTTPURLResponse *)response
{
       [self updateUI: [responseData objectForKey:[self productKeyKey]]];
        [self hideActivityIndicator];
       [[TLFCustomEvent sharedInstance] logScreenLayoutWithViewController:self];
}
```

In some cases, you need to delay triggering logging to give time for UI animations to complete or a UITableView reloadData call to complete. The Custom Event provides a method to accomplish this.

```
- (void)RESTRequestCompleted:(RESTRequest *)request responseData:(NSDictionary
*)responseData response:(NSHTTPURLResponse *)response
{
   items = [responseData objectForKey:[self itemsKey]];
   [self.itemsTable reloadData];
   [self hideActivityIndicator];
        [[TLFCustomEvent sharedInstance] logScreenLayoutWithViewController:self
andDelay:0.1];
}
```

After certain UIEvents, it may be beneficial to trigger logging, such as upon selection of an item on table view that stretches beyond one screen.

```
- (NSIndexPath *)tableView:(UITableView *)tableView willSelectRowAtIndexPath:
(NSIndexPath *)indexPath
{
      [[TLFCustomEvent sharedInstance] logScreenLayoutWithViewController:self];
      return indexPath;
}
```

A manual logging call is required to capture an alert view.

```
- (IBAction)btnSubmitFormClick:(id)sender {
       UIAlertView *alert=[[UIAlertView alloc] initWithTitle:
@"Thank You!" message:@"We will be in touch with you soon."
delegate:self cancelButtonTitle:@"Ok" otherButtonTitles: nil];
       [alert show];
       [[TLFCustomEvent sharedInstance] logScreenLayoutWithViewController:
self andRelatedViews:@[alert]];
}
```

You should also log the screen layout after the alert dialog is dismissed.

```
- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:
(NSInteger)buttonIndex
{
       [[TLFCustomEvent sharedInstance] logScreenLayoutWithViewController:self];
}
```

IBM Tealeaf screen layout logging only logs the views and controls that are on screen when the logging call is made. When `UITableView` contains more rows than can be view on a screen at once, call the screen layout logging when an item is selected. This ensures that the event matches the row selected. Use the following code in your `UITableViewDelegate` to make this change.

```
- (NSIndexPath *)tableView:(UITableView *)tableView
willSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [[TLFCustomEvent sharedInstance] logScreenLayoutWithViewController:self];
    return indexPath;
}
```

Supported controls

- UIView
- UIAlertView
- UITableView
- UITableViewCell
- UIScrollView
- UINavigationView - basic navigation bar support only
- UITabView - tab bar icons are not supported
- UIScrollView
- UICollectionView
- UILabel
- UIButton
- UITextField
- UITextView
- UIImageView
- UIActivityIndicator
- UIProgressView - custom graphics are not supported
- UISlider - custom graphics are not supported
- UISegmentedControl

Unsupported controls

- UIStepper
- UIPageControl
- UIPickerView
- UIDatePicker
- UIWebView
- MKMapView

Install Tealeaf in an Xcode project

After you download the IBM Tealeaf CX Mobile iOS Logging Framework package, you install the CX Mobile iOS Logging Framework libraries into an iOS application project.

Install process

To install Tealeaf in your Xcode project you:

1. Add required iOS frameworks to your project
2. Add Tealeaf files to your project
3. Set the Objective C linker flag

4. Add Tealeaf headers to your pch file
5. Set the UIApplication Class to use the Tealeaf Application class or modify your custom UIApplicaiton class.

Adding the required frameworks to your Xcode project

The IBM Tealeaf CX Mobile iOS Logging Framework requires a number of Apple Frameworks to function.

Before you do this task you must have downloaded the installation package and extracted it to a location on your system.

The iOS Logging Framework requires these frameworks:

- Foundation.framework
- UIKit.framework
- CoreTelephony.framework
- CoreLocation.framework
- libz.dylib
- MediaPlayer.framework
- SystemConfiguration.framework

If these frameworks are already in your project, you do not need to add them a second time.

1. In your Xcode project, select the project node in the **Project Navigator**.
2. Select your desired target under the targets list.
3. Select the **General** tab.
4. In **Linked Frameworks and Libraries**, click **+** to search and select frameworks.

Adding Tealeaf files to your Xcode project

You add the Tealeaf files to your Xcode project to add the Tealeaf library to your project.

Before you begin, you must download the installation package.

Add the Tealeaf files to the main Target.

1. Extract the `TLFLibRelease.zip` file. The **Tealeaf** folder is extracted. The **Tealeaf** folder contains all necessary files for the CX Mobile iOS Logging Framework.
2. Drag the **Tealeaf** folder onto the **Project Navigator** in Xcode. Or, in Xcode, right-click **Project Navigator** and choose **Add Files to "Your Project"...** then select the **Tealeaf** folder.
3. In the **Choose options for adding these files** dialog, check **Copy items into destination group's folder (if needed)** and **Create Groups for any added folders**.
4. Click **Finish**.

Enabling the Tealeaf framework

You must start the IBM Tealeaf CX Mobile iOS Logging Framework when you application starts.

If **DynamicConfigurationEnabled** in TLFConfigurableItems is set to `NO`, you do not have to complete this task.

1. In Xcode, locate your app delegate file.
2. Search for this code:

   ```
   -BOOL)application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
   ```
3. Add this line of code as the first line of the method application `didFinishLaunchingWithOptions`:

   ```
           [[TLFApplicationHelper sharedInstance] enableTealeafFramework];
   ```

Setting the Objective-C linker flag

Set the Objective C linker flag to load all of the Objective-C static libraries.

1. In Xcode, go to the **Build Settings** for your project and select the main Target.
2. In the search box, enter `Other Linker Flags`.
3. If the `-ObjC` flag is not listed as an `Other Linker Flags` entry, double-click the row and select the +, then enter `-ObjC`. If the `-ObjC` flag is listed, you are done with this task.

Adding Tealeaf Headers to the pch file

Adding Tealeaf Headers to the pch file lets you use other IBM Tealeaf functionality without adding the headers to each file.

1. In Xcode, locate your pch file by typing `Option + Command + J` to open the **Project Navigator** search box. Then, type `<ProjectName>-Prefix.pch` in the **Project Navigator** search box.
2. Open your pch file.
3. Search for the `#ifdef_OBJC_block`.
4. Copy this code into the `#ifdef_OBJC_block`:

   ```
           #import "TLFPublicDefinitions.h"
           #import "TLFApplication.h"
           #import "TLFCustomEvent.h"
           #import "TLFApplicationHelper.h"
   ```
5. Save and exit the pch file.

Modify your application to use Tealaf classes

The changes that are needed in the code depend on whether your project implements a custom UIApplication Class.

No custom UIApplication class

If your application does not have its own custom UIApplication class, you can use the IBM Tealeaf UIApplication class.

Custom UIApplication class

If your application has its own custom UIApplication class (for example, named CustomerUIApplication), but there are no sendAction and sendEvent methods in CustomerUIApplication class, review the following information.

If your application has its own custom UIApplication class (for example, named CustomerUIApplication), and there are sendAction and sendEvent methods in CustomerUIApplication class, you can add code to point to the Tealeaf sendAction and sendEvent classes.

Using the TLFApplication class

If your application does not have its own custom UIApplication class, it can use the IBM Tealeaf CX Mobile iOS Logging Framework TLFApplication class. In your application's `main.m` file, you must tell UIApplicationMain to use the IBM Tealeaf subclass of UIApplication.

1. In your `main.m` class, search for code that uses the UIApplication:

   ```
           return UIApplicationMain(argc, argv, nil, NSStringFromClass
   ([AppDelegate class]));
   ```

2. Replace the third argument, `nil`, with the name of the `TLFApplication class`. The line should now look like;

   ```
           return UIApplicationMain(argc, argv, NSStringFromClass
   ([TLFApplication class]), NSStringFromClass([AppDelegate class]));
   ```

Modifying the custom UIApplication class

Modify your custom UIApplication class. If your custom class does not have sendAction and sendEvent methods, you can add them. If your custom class does have sendAction and sendEvent methods, you can modify them to point to the Tealeaf methods. The Tealeaf methods have logging built into them.

1. Optional: If your `UIApplication` class does not have `sendAction` and `sendEvent` classes, add the `sendAction` and `sendEvent` methods to your custom `UIApplication` class. For example:

   ```
   @implementation CustomerUIApplication
              - (void)sendEvent:(UIEvent *)event
             {
                 [[TLFApplicationHelper sharedInstance] sendEvent:event];
                 [super sendEvent:event];
             }
             - (BOOL)sendAction:(SEL)action to:(id)target from:(id)sender
   forEvent:
             (UIEvent *)event
             {
                 [[TLFApplicationHelper sharedInstance] sendAction:action
   to:target
               from:sender forEvent:event];
              return [super sendAction:action to:target from:sender
   forEvent:event];
             }
   ```

2. Optional: If your custom `UIApplication` class has `sendAction` and `sendEvent` methods, modify the methods to point to Tealeaf `sendAction` and `sendEvent`. For example, add this line to the `sendEvent` method:

   ```
   [[TLFApplicationHelper sharedInstance] sendEvent:event];
   ```

   For example, add this line to the `sendAction` method:

   ```
   [[TLFApplicationHelper sharedInstance] sendAction:action to:target
   from:sender forEvent:event];
   ```

Configure Tealeaf

You configure several items for your application in Tealeaf, including how screen layouts are logged, Target page location, kill switch location, and whether gestures will be logged.

Configurable items

In Tealeaf, you configure:

- How screen layouts are logged.

- The Target page URL.
- The kill switch URL.
- Auto-instrumentation

TLFConfigurableItems.plist

Everything that you configure is in the `TLFConfigurableItems.plist` file. This file is in the Install Package in the `Tealeaf/Resources/TLFResources.bundle` file.

How screen layouts are logged

Tealeaf can log screen images as base64 or as MD5 checksum with png or jpg images. Set `GetImageDataOnScreenLayout` to `YES` to capture base 64 data. Set `GetImageDataOnScreenLayout` to `NO` to log MD5 checksum and png or jpg images. This option creates smaller payloads in production and is the recommended setting.

Set Target URL

All events that are captured are sent in JSON format to a Target page. The Target page acknowledges the receipt of the JSON message and forwards the client-side events to Tealeaf. The person that sets up Tealeaf on the server creates the Target page. The Target page is set with the `PostMessageUrl` field.

Set the kill switch URL

The Kill Switch is used to control logging. When the kill switch is enabled, it must have a URL to check before the framework initializes. When the page is reachable, the framework initializes. If the page is not reachable, because of network problems or because you disabled it on your server, the framework does not initialize. The kill switch URL is set by the person who sets up Tealeaf on the server. The kill switch URL is set with the`KillSwitchUrl` field.

Auto-instrumentation

By default, the CX Mobile iOS Logging Framework automatically instruments your application for a set of predefined events. You can disable auto-instrumentation and then apply custom instrumentation for elements in your application. Auto-instrumentation is set with the `DisableAutoInstrumentation` field. You should leave this setting as `YES`.

Configuring Tealeaf for your application

You configure Tealeaf to use specific URLS for logging events and to control message flow, and set how screen layouts are logged.

All of the configuration in this task involves modifying settings in the `TLFConfigurableItems.plist` file in the Tealeaf Resources folder that you added to your Xcode project.

1. In your project in Xcode, open the `TLFConfigurableItems.plist` file.
2. Set the `GetImageDataOnScreenLayout` to `NO`.
3. Set the `PostMessageUrl` to the URL of the Target page for your app.
4. Set the `KillSwitchUrl` to the URL for the kill switch for your app.
5. Save and exit the `TLFConfigurableItems.plist` file.

Configure gesture capture

Tealeaf provides a module log user gestures in your application. Tealeaf captures several gestures. If you are using your own gestures in your application, you modify the delegate for your Gesture Recognizer to work with the Tealeaf capture function.

Process

To configure gestures for your application:

1. Modify the `TLFConfigurableItems.plist` file and set the `SetGestureDetector` field to `YES` to log gestures.
2. If you are using your own gestures, modify the delegate for your Gesture Recognizer to work with Tealeaf Capture.

Log gestures

You can capture gestures that the user makes on your application. Gesture capture is set with the `SetGestureDetector` field. Gestures are logged as Type 11 JSON messages. The captured gestures include:

- Tap
- Tap and Hold
- Double-tap
- Swipe in any direction
- Swipe up
- Swipe down
- Swipe left
- Swipe right
- Pinch
- Spread

Custom gestures and Tealeaf capture

You might have your own gestures in your application. For Tealeaf to log the gestures, you need to modify the delegate for your Gesture Recognizer. You only need to do this if you are using your own gestures in your application. If you are using the Tealeaf gestures, you do not need to do this.

Gesture events captured

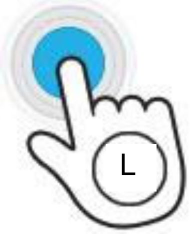
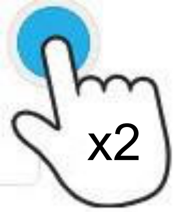
Gestures that are used to select items in an application or to adjust views in the application are captured by Tealeaf.

Tap gestures

This table lists and describes the tap gestures that are captured from web and mobile apps:

Note: The arrows that illustrate the direction of a swipe or pinch gesture are not supported by the Internet Explorer browser.

Table 3. Tap gestures

Gesture name	Description	Image displayed in Replay
Tap	This gesture is a one-finger gesture. For a tap gesture, one-finger taps and lifts from the screen in 1 location.	
Tap and Hold	This gesture is a one-finger gesture. For a Tap and Hold gesture, one-finger presses and stays on the screen until information is displayed or an action occurs. **Note:** The response to a Tap and Hold gesture can vary from one application to another. For example, a Tap and Hold gesture might display an information bubble, magnify content under the finger, or present the user with a context menu.	
Double tap	This gesture is a one-finger gesture. For a double tap gesture, one-finger taps twice in close succession in 1 location of the screen.	

Swipe gestures

This table lists and describes the swipe gestures that are captured from web and mobile apps:

Table 4. Swipe gestures

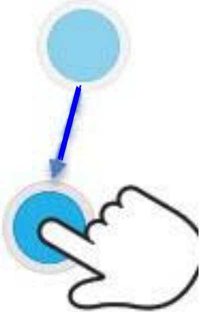
Gesture name	Description	Image displayed in Replay
Swipe vertically	This gesture is a one-finger gesture. For a swipe vertically gesture, one-finger: 1. taps and holds in 1 location of screen, 2. continues to engage screen while it moves up or down 3. lifts from the screen in different location. **Note:** The initial tap becomes lighter in color, while the destination is highlighted by a darker color	

Table 4. Swipe gestures (continued)

Gesture name	Description	Image displayed in Replay
Swipe horizontally	This gesture is a one-finger gesture. For a swipe horizontally gesture, one-finger: 1. taps and holds in 1 location of screen, 2. continues to engage screen while it moves left or right 3. lifts from the screen in different location. **Note:** The initial tap becomes lighter in color, while the destination is highlighted by a darker color	

Resize gestures

This table lists and describes the resize gestures that are captured from web and mobile apps:

Note: See the *IBM Tealeaf Customer Experience 9.0.1 Release Notes* for information about a known limitation for handling some iOS pinch gestures.

Table 5. Resize gestures

Gesture name	Description	Image displayed in Replay
Pinch open	Sometimes referred to as a *spread* gesture, this is a two-finger gesture. For a pinch open gesture, 2 fingers: 1. tap and hold in 1 location of the screen, 2. maintain contact with the screen while the fingers move apart from each other in any direction, 3. lift from the screen at a new location.	 **Note:** Accompanying arrows indicate the direction (open or close) of the pinch
Pinch close	This gesture is a two-finger gesture. For a pinch close resize gesture, 2 fingers: 1. tap and hold in 1 location on the screen, 2. maintain contact with the screen while the fingers move toward each other, 3. lift from the screen at a new location.	 **Note:** Accompanying arrows indicate the direction (open or close) of the pinch

Configuring Gesture capture for your application

You modify the `TLFConfigurableItems.plist` file to enable gesture capture for your application.

All of the configuration in this task involves modifying settings in the `TLFConfigurableItems.plist` file in the Tealeaf Resources folder that you added to your Xcode project.

1. In your project in Xcode, open the `TLFConfigurableItems.plist` file.
2. Set the `SetGestureDetector` field to `YES` to log gestures.
3. Save and exit the `TLFConfigurableItems.plist` file.

Modifying the delegate for your Gesture Recognizer to work with Tealeaf capture

If you have your own gestures recognizer in your application, the code might affect the Tealeaf gesture capture feature. To ensure that your gestures and Tealeaf capture work together, you add a method to the delegate for your Gesture Recognizer. You do this task only if you are using your own gesture recognizer.

1. Locate the delegate for your Gesture Recognizer.
2. Add this method to the delegate:

```
- (BOOL)gestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
shouldRecognizeSimultaneouslyWithGestureRecognizer:
(UIGestureRecognizer *)otherGestureRecognizer
{
    return YES;
}
```

Log Exceptions

Exceptions are the way that a system or framework communicates to the application that something has gone wrong and not to continue with the execution of the program unless the exception is one of the expected ones. You can manually and automatically log caught exceptions using the Tealeaf SDKs so that the exception information can be used for analytics.

Three ways to log exceptions

In iOS SDK there are three ways to log exceptions that are trapped by your application exception handler. These methods do not use the Cocoa SDK, which is not exception- safe. This table lists the methods used to log exceptions and the parameters used in each method:

Method	Parameters
`- (BOOL)logNSExceptionEvent: (NSException *)exception;`	Where: • @param exception - The caught NSException instance. • @return if the event was successfully logged or not.
`- (BOOL)logNSExceptionEvent: (NSException *)exception dataDictionary:(NSDictionary*) dataDictionary;` You set the `NSSetUncaughtExceptionHandler` of your AppDelegate.m inside of: `- (BOOL)application: (UIApplication *)application didFinishLaunchingWithOptions: (NSDictionary *)launchOptions`	Where: • @param exception - The caught NSException instance. • @param dataDictionary - Additional data about the exception. • @return if the event was successfully logged or not.

Method	Parameters
`- (BOOL)logNSExceptionEvent: (NSException *)exception dataDictionary:(NSDictionary*) dataDictionary isUnhandled:(BOOL) unhandled;`	Where: • @param exception - The caught NSException instance. • @param dataDictionary - Additional data about the exception. • @param unhandled - Indicates whether the exception was caught by an exception handler or not. • @return if the event was successfully logged or not.

Example

In this example, you have a method that causes an exception:

```
-(void)aMethod {
 [self causesAnException];
}
```

You add an `@try` , `@catch`, and the `[[TLFCustomEvent sharedInstance] logNSExceptionEvent:exception];` method to handle the exception:

```
-(void)aMethod {
 @try {
  [self causesAnException];
 }
 @catch(NSException *exception) {
 [[TLFCustomEvent sharedInstance] logNSExceptionEvent:exception];
}
}
```

Log uncaught exceptions

You can log uncaught exceptions by setting up and adding an `NSUncaughtExceptionHandler`.

Logging exceptions

Use the examples in this task as a guide to adding exception logging to your application.

You might want to use the top-level `NSSetUncaughtExceptionHandler( &SampleAutoUncaughtExceptionHandler);` to identify bugs in your application.

The current iOS SDK catches some exceptions and prevents them from being logged to the target page. In some cases this may prevent an application from catching the exception.

1. Determine the method for which you want to log exceptions. For example, you have a method:

   ```
   -(void)aMethod {
    [self causesAnException];
   }
   ```

2. Optional: Add the exception method that you want to use to the method for which you want to

   Add `@try` , `@catch`, and the `[[TLFCustomEvent sharedInstance] logNSExceptionEvent:exception];` method to handle the exception:

```
-(void)aMethod {
 @try {
  [self causesAnException];
 }
 @catch(NSException *exception) {
 [[TLFCustomEvent sharedInstance] logNSExceptionEvent:exception];
}
}
```

3. Optional: Set up an `NSUncaughtExceptionHandler` for logging for uncaught exceptions: For example:

```
void SampleAutoUncaughtExceptionHandler(NSException *exception) {
 [[TLFCustomEvent sharedInstance] logNSExceptionEvent:exception];
}
NSSetUncaughtExceptionHandler(&SampleAutoUncaughtExceptionHandler);
```

Configure DOM Capture

DOM Capture is an alternative to traditional UI Capture and Replay. DOM Capture is used to capture anything that is not exposed in response HTML. DOM Capture configuration is part of the Configuration wizard that you run to configure UI Capture.

Process

DOM Capture is part of the Replay module. To enable DOM Capture, you must enable the Replay module. When you configure DOM Capture, you use the Configuration wizard to:

1. Enable DOM Capture
2. Configure user interaction triggers for DOM Capture (for example, screenview load or user clicks).
3. Configure maximum threshold size for the DOM Capture message.

Limitations

DOM Capture operates on a page level.

Privacy Rules that specify regular expressions as identifiers are currently not supported for this release.

DOM Capture can be replayed only in BBR.

Hybrid application and Native applications

How you configure DOM Capture varies based on how you are using DOM Capture.

IF you are using DOM Capture for...	THEN you...
which PCA cannot be used to listen to request and responses for Native applications	1. Install the UIC library in your application. 2. Modify either the: • `defaultconfiguration.js` file in the UIC library to enable the library to collect a DOM Capture JSON object. • Native application to fire DOM Capture. If the HTML page in the webview does not fire on page load or maybe the page changes dramatically you need to fire DOM capture from within your application. You do not use the Configuration wizard for DOM Capture in Native applications for situations in which PCA cannot be used.
Hybrid applications that use WebView in either iOS hybrid applications or Android hybrid applications	1. Use the Configuration wizard to do basic configuration and enable DOM Capture. 2. Add `.domcapture` to the events in your application for which you want to use DOM Capture. You can use DOM Capture in click, change, load, and unload events. In the event you can specify: • targets for the event • screenview names (for load and unload events only) • A delay in milliseconds for the DOC Capture to wait until the snapshot is taken.

Configuring DOM Capture and Replay for Native iOS applications that cannot use PCA

You configure DOM capture for a Native iOS application that cannot use PCA by modifying the `defaultconfiguration.js` file. If the HTML page in the webview does not fire on page load or if the page changes dramatically, you need to fire DOM capture from within your Native iOS application.

Before you do this task you must install the UIC library in your native application. All of the modifications that you make are in your Native iOS application.

1. Implement these methods in the `UIWebViewDelegate`:

```
(void)webViewDidFinishLoad:(UIWebView *)webView {

  [[TLFCustomEvent sharedInstance] logScreenLayoutWithViewController:self];

}


- (BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:
(NSURLRequest *)request navigationType:
(UIWebViewNavigationType)navigationType {

    return YES;

}
```

2. Modify the `defaultconfiguration.js` file and set the DOM Capture options that you want to use:

```
replay: {
    // DOM Capture configuration
    domCapture: {
        enabled: true,
        // Filter object for matching rules is similar to the Privacy
configuration
        // It accepts a mandatory "event" followed by one or more
optional targets
        // as well as an optional delay after which to take the DOM snapshot.
        triggers: [
            {
                event: "load"
            }
        ],
        // DOM Capture options
        options: {
            captureFrames: true,   // Should child frames/iframes be
captured
            removeScripts: true    // Should script tags be removed from
the captured snapshot
        }
    }
}
```

3. If DOM Capture does not fire on load, set DOM Capture to fire from your application by adding this code to your native iOS application for the screenview that you want to capture:

```
if (TLT === undefined) {
    console.log('TLT is undefined!');
} else {
    if (TLT.logScreenviewLoad === undefined) {
        console.log('Could not invoke TLT.logScreenviewLoad API!');
    } else {
        TLT.logScreenviewLoad("root");
        console.log('logScreenviewLoad:');
    }

    if (TLT.logDOMCapture === undefined) {
        console.log('Could not invoke TLT.logDOMCapture API!');
    } else {
        dcid = TLT.logDOMCapture(window.document, {});
        console.log('logDOMCapture:' + dcid);
    }
}
```

4. Optional: If the webview has images that are slow to load, you can add a delay to the DOM Capture with this method:

```
-(void)webViewDidFinishLoad:(UIWebView *)webView {

    [[TLFCustomEvent sharedInstance] logScreenLayoutWithViewController:self
andDelay:0.2];

}
```

Integrate Tealeaf and Worklight

Worklight® is IBM's Mobile First Platform for developing both Hybrid and Native Apps on multiple mobile platforms. For logging activities on your application, you might want to integrate the Tealeaf library inside of a Worklight "Hybrid" application. Worklight provides an Eclipse plug-in called "Worklight Developer Studio" to help Developers create Mobile Apps more productively.

Development environment

To integrate Tealeaf with Worklight, you need these files:

- Eclipse IDE for Java™ EE Developers (Kepler): http://www.eclipse.org/downloads/packages/release/Kepler/SR2
- Worklight Developer Studio version 6.1. You need the compressed file `iws_update_site_wde.6.1.0.2.zip`

You must also install the:

1. Worklight Developer Studio inside Eclipse following the instructions in the Worklight documentation.
2. Android ADT plug-in in your Eclipse instance.

Worklight high-level single project

Within Worklight, you can create and manage Mobile project artifacts in a single, high-level project called "Worklight Project". Artifacts include server-side adapters, multiple android projects, and multiple iOS projects All artifacts in the single, high-level project have access to the same resources..

Differences between Worklight 6.1 and Worklight 6.2

In 6.1, Worklight and Tealeaf are packaged together. In Worklight 6.2 they are no longer packaged together. For 6.2, you must do additional steps to integrate the two products.

Modify Tealeaf and Worklight classes

Part of integrating Tealeaf and Worklight 6.2 is extending and modfying Tealeaf and Worklight classes. This table lists the classes and methods that you modify and shows examples of the modifications:

Method or class	Example

OtherSources/main.m	`#import <UIKit/UIKit.h>`
	`#import "TLFApplication.h"`
	`int main(int argc, char *argv[]) {`
	`@autoreleasepool {`
	`int retVal = UIApplicationMain(argc,`
	`argv, NSStringFromClass([TLFApplication`
	`class]),`
	`@"MyAppDelegate");`
	`return retVal;`
	`}`
	`}`

Method or class	Example

Classes/HelloWorklight.m	- (BOOL)application:(UIApplication
	*)application
	didFinishLaunchingWithOptions:
	(NSDictionary
	*)launchOptions
	{
	BOOL result = [super application:
	application
	didFinishLaunchingWithOptions
	:launchOptions];
	// A root view controller must be
	created in application:
	didFinishLaunchingWithOptions:
	self.window = [[UIWindow alloc]
	initWithFrame:[[UIScreen mainScreen]
	bounds]];
	UIViewController* rootViewController
	= [[Compatibility50ViewController
	alloc] init];
	[self.window setRootViewController:
	rootViewController];
	[self.window makeKeyAndVisible];
	[[WL sharedInstance] showSplashScreen];
	// By default splash screen will be
	automatically hidden once Worklight
	JavaScript framework is complete.
	// To override this behaviour set
	autoHideSplash property in
	initOptions.js to false and use
	WL.App.hideSplashScreen() API.
	[[WL sharedInstance]
	initializeWebFrameworkWithDelegate
	:self];
	[[TLFApplicationHelper sharedInstance]
	enableTealeafFramework];
	return result;
	}

Process

To integrate Tealeaf and Worklight, you:

1. Create a high-level Worklight project called "Worklight Project"
2. Add the Tealeaf SDK to the high-level "Worklight Project".
3. Create an iOS project under the high-level "Worklight Project"
4. Convert the project to Xcode.
5. Modify iOS classes and methods (Integrating with Worklight 6.2 only)

Creating and configuring the high-level Worklight project

You can manage your Tealeaf and Worklight integration with the high-level Worklight Project. To integrate Tealeaf and Worklight 6.1, you create the high-level Worklight project, add the Tealeaf SDK to the project, and activate Tealeaf in the JavaScript layer.

For Worklight 6,2, you must modify and create classes for integrating Tealeaf and Worklight. Create these files before you begin this task.

In this task, you work in the Eclipse environment then convert the project to XCode. You modify your application and add libraries to the project.

1. Create the high-level Worklight project:
   a. In Eclipse, select **New** > **Project** > **Worklight Project**.
   b. Enter the name of the project, for example `Worklight Project` and select **Hybrid Application**.
   c. Enter the name of the Hybrid Application that you are creating. For example, `HelloWorklight`. The high-level project is created and a Hybrid application that is called `HelloWorklight` is in the `apps` folder.
2. Activate the Tealeaf SDK on the high-level Worklight project:
   a. Copy the `configuretealeaf.js` file to the `apps/HelloWorklight/iphone/js` folder.
3. Create an iphone project under the high-level Worklight Project:
   a. Right click on the `HelloWorklight` folder under **Apps**.
   b. Select **New** > **Worklight Environment**.
   c. Select **iphone**.
4. Convert the project to Xcode. Select **Run As** > **XCode Project**
5. Optional: For integrating Tealeaf and Worklight 6.2 only: Modify the classes required for Tealeaf and Worklight integration:
   a. Modify Other Sources/main.m.
   b. Modify the Classes/HelloWorklight.m.

Quick start for server configuration

This section describes the basic steps to configure the IBM Tealeaf CX Passive Capture Application and Windows based servers to capture and process data that is submitted from the CX Mobile iOS Logging Framework.

To enable processing of submitted data, complete the steps in the following sections.

Data privacy

IBM Tealeaf provides mechanisms for masking or blocking sensitive customer information, such as credit card numbers, from being transmitted and captured by IBM Tealeaf.

Through the CX Mobile iOS Logging Framework, you can specify the fields that must be blocked or masked in your web application. When applied, data privacy ensures that these data elements are never transmitted to IBM Tealeaf.

Note: Due to the way in which client framework data is submitted to IBM Tealeaf for capture, to mask or block sensitive data you apply filtering through the capturing client framework. While other IBM Tealeaf features to manage data privacy can be deployed, they are not easy to implement on the format of data captured from the client frameworks.

- See "Data Privacy in IBM Tealeaf Client Frameworks" in the *IBM Tealeaf Client Framework Data Integration Guide*.
- For more information about handling sensitive data in general, see "Managing Data Privacy in IBM Tealeaf CX" in the *IBM Tealeaf CX Installation Manual*.

Target page for traffic capture

IBM Tealeaf is designed to capture traffic between a client and a web server. To facilitate capture, you add a target page to your web server environment to which the CX Mobile iOS Logging Framework can submit posts.

You can use the same target page that is available for IBM Tealeaf CX UI Capture for AJAX. See "IBM Tealeaf target page" in the *IBM Tealeaf CX UI Capture for AJAX Guide*.

After you add the target page to your web environment and enable the appropriate access permissions, you must configure the URL for the target page in the `TLFConfigurableItems.plist` page.

Note: If needed, you can configure the client framework to submit by HTTPS by adding the protocol identifier to the post URL.

Traffic volume management

You can add a sampling function to work with the CX Mobile iOS Logging Framework kill switch. This sampling function can be used to throttle the sampling rate and thus the volume of traffic that is forwarded for capture.

For more information about sampling functions for various server environments, see Chapter 6, “Sample code,” on page 91.

Implementing screenViews

For pages in which the state or context can be switched without re-rendering the page, IBM Tealeaf segments the data between states by using an object that is called a screenView.

For example, if a page contains multiple tabs in it, each of which represents a different stage in a checkout process, you instrument each tab in the page as a distinct screenView.

To implement a screenView for a page, complete the following steps.

1. `logicalPageName` for a screenView is the current `UIViewController's` classname or title.
2. If the prior step is not completed, call `[TLFCustomEvent sharedInstance] logAppContext` and pass the `logicalPageName`. For example:

   ```
   [[TLFCustomEvent sharedInstance] logAppContext:logicalPageName
    applicationContext:applicationContext referrer:referrer] ;
   ```

Traffic capture configuration on the CX Passive Capture Application

Data is submitted from the CX Mobile iOS Logging Framework to the CX Passive Capture Application by using specific content types.

The CX Passive Capture Application is typically configured to capture these content types by default. You verify that these content types are enabled for capture through the CX Passive Capture Application web console.

Note: After the completion of the steps in this section, data is processed by IBM Tealeaf.

Verifying CX Passive Capture Application capture type configuration

You use the CX Passive Capture Application web console to verify that the content types submitted by the CX Mobile iOS Logging Framework are being captured by the CX Passive Capture Application.

Note: Depending on the version of the CX Passive Capture Application that you installed, the required content types may already be configured for capture.

The CX Mobile iOS Logging Framework submits messages by using the `application/json` content type.

Note: Each IBM Tealeaf CX Mobile iOS Logging Framework can use a different content type for submitting events for capture to IBM Tealeaf. Be sure to review and verify the content type for each deployed client framework.

1. Log in to the CX Passive Capture Application web console.
   ```
   <PCAServer>:8080
   ```
   where `<PCAServer>` is the host name of the CX Passive Capture Application server.
2. Click the **Pipeline** tab.
3. Click **Edit Type Lists**.
4. In the **Capture All POST Types** box, verify that the following values are included.
   ```
   text/json
   text/x-json
   application/json
   application/x-json
   ```
5. Click **Add**.
6. The CX Passive Capture Application is now configured to capture the required content types. All subsequent hits of this type are captured.
7. Save your changes.
   - See "PCA Web Console - Pipeline Tab" in the *IBM Tealeaf CX Passive Capture Application Manual*.

Configuring CX Passive Capture Application for screen capture from CX Mobile iOS Logging Framework

Optionally, you can set up the CX Mobile iOS Logging Framework to do a screen capture during the initial load of each view or screen of your web application. These screen captures are forwarded to the IBM Tealeaf Target Page in `PNG` and `JPG` format for use during session display.

`PNG` files are not compressed, while `JPG` is a compressed format. A`PNG` file is approximately 20 KB to 35 KB in size; a `JPG` file is 6 KB to 15 KB.

When this option is enabled, you must configure the CX Passive Capture Application to capture these screens. By default, the CX Passive Capture Application drops capture of binary or static content, so you must configure it to capture images that are submitted as binary POSTs to the target page. See “Screen capture at run time” on page 77.

1. Log in to the CX Passive Capture Application web console.
   ```
   <PCAServer>:8080
   ```
   Where `<PCAServer>` is the host name of the CX Passive Capture Application server.

2. Click the **Pipeline** tab.
3. Click **Edit Type Lists**.
4. In the **Excluded File Extensions** list, verify that `png` or `jpg` is listed.
5. In the **Included File Extensions** list, verify that `png` or `jpg` is not listed.

   Note: If a file extension is included in this list, then all instances that are sent as responses are captured, which greatly expands the volume of data that is captured by the CX Passive Capture Application. Capture in this manner is not required.
6. In the **Binary POST Types** box, enter the following value.

   ```
   image/png
   ```
7. Click **Add**.
8. The `image/png` POST type is added and enabled for capture. This setting allows the PNG posts to be captured by the CX Passive Capture Application.
9. Save your changes.

See "PCA Web Console - Pipeline Tab" in the *IBM Tealeaf CX Passive Capture Application Manual*.

Enabling decompression of compressed POSTs

The CX Mobile iOS Logging Framework automatically compresses POST data. You must configure the CX Passive Capture Application to decompress them.

1. In the CX Passive Capture Application Web Console, click the **Pipeline** tab.
2. Select **Inflate compressed requests and responses**.
3. Save your changes.

The compressed POSTs are now automatically decompressed by the CX Passive Capture Application and processed normally.

Options for monitoring captures and processing

You use different tools for testing your configuration and monitoring captures on an ongoing basis.

At target page

You can test the basic functionality of the target page by triggering GET and POST actions on the URL where the target page was installed.

See "Unit tests of target page" in the *IBM Tealeaf UI Capture for Ajax Guide*.

In Windows pipeline

You can monitor the capture and processing of hits in the Windows pipeline in real time through the IBM Tealeaf Management System. See "TMS Pipeline Status Tab" in the *IBM Tealeaf cxImpact Administration Manual*.

Sessionization for iOS applications

The CX Mobile iOS Logging Framework uses a tiered approach to generating identifiers for mobile native application sessions. A summary of the approaches for generating identifiers follows.

- Use `TLTSID` identifier that is provided by web server.

This solution uses the session identifier that is provided by your web server environment, which forces the mobile native application to use identifiers that are consistent with your non-mobile sessions. Ideally, this identifier is provided as a TLTSID value, which is the default session identifier value within IBM Tealeaf.

Important: If possible, use this method of generating the session identifier.

To enable this method of generating session identifiers, the first hit of your mobile native application session must be forced to be a web hit that touches the server or servers that generate session identifiers.

Ideally, the session identifier that is generated by your web server is provided by the IBM Tealeaf Cookie Injector, which generates session IDs that are unique within IBM Tealeaf. See "Installing and Configuring the Tealeaf Cookie Injector" in the *IBM Tealeaf Cookie Injector Manual*.

- Use another identifier that is provided by web server

  In some environments, the TLTSID value is not used as the session identifier. In these cases, you must force the first hit to be a web hit targeting the web server, and you must deploy a session agent in your Windows pipeline to map the proper session identifier for IBM Tealeaf.

  Important: This method is not validated in a customer environment and is not officially supported. For more information, contact IBM technical support.

  To enable this method of generating session identifiers, the first hit of your mobile native application session must be forced to be a web hit that touches the server or servers that generate session identifiers.

  If you are using a session identifier other than TLTSID, you must include the Sessioning session agent in your pipeline to identify your session identifier for IBM Tealeaf. If you already deployed IBM Tealeaf to capture non-mobile sessions and the session identifier was already defined by your web server, this configuration was probably already completed. Verify that it is present and functioning in the Windows pipeline. See "Sessioning Session Agent" in the *IBM Tealeaf CX Configuration Manual*.

- Configure the TLTSID by changing the string value of SessionizationCookieName from TLFConfigurableItems.plist. SessionTimeout should be set together with SessionizationCookieName. When the time out happens, CX Mobile iOS Logging Framework auto generates a new Session ID and assigned it to the variable of SessionizationCookieName. This customized session identifier is a hashed value that is submitted as a cookie in the first hit and all subsequent hits.

  To get the generated session ID, implemented the following:

  ```
  @protocol TLFLibDelegate <NSObject>
  @optional /** After set a delegate to your TLFApplication implement this
  callback to generate your custom Session ID */
  - (NSString*)sessionIdGeneration; @end
  ```

  If you do not configure SessionizationCookieName, by default, it will use TLTSID, which is generated by Sessioning session agent in your pipeline.

IBM Tealeaf CX provides multiple mechanisms for sessionization. See "Managing Data Sessionization in Tealeaf CX" in the *IBM Tealeaf CX Installation Manual*.

A note on sessionization for upgraded environments

If you upgraded your CX Mobile iOS Logging Framework from a version before 8.6.7.3, the method of sessionization changed.

- Previously, the CX Mobile iOS Logging Framework submitted session identifiers using the `X-Tealeaf-Session` header.
- Beginning in iOS 5, the headers are no longer available to the local application.
- To sessionize, IBM Tealeaf now submits session identifiers as cookies.

After you upgrade your CX Mobile iOS Logging Framework from a version before 8.6.7.3, you change how sessionization is managed within your native application.

- If you use the `TLTSID` value, you do not need the Sessioning session agent to map session identifiers into the request.
- The CX Mobile Android Logging Framework uses the Sessioning session agent for session identification. Do not remove it if you are also deploying an Android mobile native application. See "Tealeaf Android Logging Framework Installation and Implementation" in the *IBM Tealeaf CX Mobile Android Logging Framework Guide*.

Runtime configuration

As needed, you can change framework settings within the client application during run time. You define these settings during initialization of the application by using the framework API, and update them as needed.

The following configuration items can be configured dynamically from the client.

- Dynamic PostMessageURL: Changes the target URL for iOS Logging Framework as needed.
- KillSwitchURL: Activates the killswitch on the iOS Logging Framework as needed.

See "Dynamic configuration items" on page 76.

IBM Tealeaf events for CX Mobile iOS Logging Framework

The JSON format is used to track data that is captured by the CX Mobile iOS Logging Framework.

Data type
Description

Client Framework data (JSON)
If you are using step-based eventing, data from the client framework is submitted in JSON format and is available through browser based replay for review and eventing. See "Step-Based Eventing" in the *IBM Tealeaf Event Manager Manual*.

For a walkthrough of how to capture this data into IBM Tealeaf events, see "Integrating Client Framework Data into Tealeaf" in the *IBM Tealeaf Client Framework Data Integration Guide*.

Client Framework data (hit-splitting)
Legacy method. See "Client framework versions supported in this documentation" on page 5.

JSON message type schemas and examples

JSON messages are categorized by type for processing. Tealeaf supports 12 JSON message types.

This table lists and describes the supported JSON message types:

Table 6. Schema by Message Type

Type	Message Type	Description
1	**"Client state (Type 1) messages" on page 32**	Any object that shows the current state of client.
2	**"ScreenView (Type 2) messages" on page 34**	Any message that indicates changes in view on the "screen". The "screen" is the page, view, or activity where the visitor is in the application.
3	**"Connections (Type 3) messages" on page 36**	Any request or response that the application performs during capture.
4	**"Control (Type 4) messages" on page 37**	User interface control that fires an event to which Tealeaf listens for capture.
5	**"Custom Event (Type 5) messages" on page 40**	Any custom log event from any location in application.
6	**"Exception (Type 6) messages" on page 41**	Any exception that the application can throw.
7	**"Performance (Type 7) messages" on page 42**	Performance data from a browser.
8	**"Web Storage (Type 8) messages" on page 43**	Any object that contains information about local storage information on the browser.
9	**"Overstat Hover Event (Type 9) messages" on page 43**	Any object that contains information about mouse hover and hover-to-click activity.
10	**"Layout (Type 10) messages" on page 44**	Any message that shows the current display layout of a native page.
11	**"Gesture (Type 11) messages" on page 46**	Any message that shows a gesture that fires a higher touch event that Tealeaf listens to for capture.
12	**"DOM Capture (Type 12) message example" on page 53**	Any object that contains serialized HTML data (DOM snapshot) of the page.

Message header properties

All messages contain message header properties consisting of two properties that contain the message type and the time that is offset from the start of the session in milliseconds.

Note: All time measurements in the JSON object schema are in milliseconds.

Message header properties schema

```
"offset": {
    "title": "Milliseconds offset from start of stream",
    "type": "integer",
    "required": true
},"screenViewOffset": {
    "title": "Milliseconds offset from start of ScreenView",
    "type": "integer",
    "required": true
},"count": {
    "title": "The number of the message being sent",
    "type": "integer",
    "required": only used for UIC
},"fromWeb": {
    "title": "Used to identify if it came from Web or Native application",
    "type": "boolean",
    "required": true
},"type": {
    "title": "Message header type",
    "type": [ {
        "enum": [1],
        description: "CLIENT_STATE"
        },
        "enum": [2],
        description: "APPLICATION_CONTEXT"
        }],
        "enum": [3],
        description: "CONNECTION"
        },
        "enum": [4],
        description: "CONTROL"
        },
        "enum": [5],
        description: "CUSTOM_EVENT"
        }],
        "enum": [6],
        description: "EXCEPTION"
        }],
    "required": true
},
```

Message header properties schema

```
"offset": {
    "title": "Milliseconds offset from start of stream",
    "type": "integer",
    "required": true
},"screenViewOffset": {
    "title": "Milliseconds offset from start of ScreenView",
    "type": "integer",
    "required": true
},"count": {
    "title": "The number of the message being sent",
    "type": "integer",
    "required": only used for UIC
},"fromWeb": {
    "title": "Used to identify if it came from Web or Native application",
    "type": "boolean",
    "required": true
},"type": {
    "title": "Message header type",
    "type": [ {
        "enum": [1],
        description: "CLIENT_STATE"
        },
        "enum": [2],
        description: "APPLICATION_CONTEXT"
        }],
        "enum": [3],
        description: "CONNECTION"
        },
        "enum": [4],
        description: "CONTROL"
        },
        "enum": [5],
        description: "CUSTOM_EVENT"
        }],
        "enum": [6],
        description: "EXCEPTION"
        }],
    "required": true
},
```

Client state (Type 1) messages

Client state messages are delivered on a schedule basis or on changes to the environment state on the client. These are Type 1 JSON messages.

Note: Replay of client state messages is not supported, except for scroll events. Replay of scroll events that are captured from the client is supported for mobile sessions only in BBR only. See *Search and Replay for Mobile Web*.

Client State (Type 1) message schema

This is the schema for the Client State (Type 1) messages.

```
{
    "$ref" : "MessageHeader",
    "mobileState": {
        "description": "Logical page being loaded for iOS and Android",
        "type": "object",
        "properties": {
            "orientation": {
                "title": "Current orientation of the device",
                "type": "integer",
                "required": true
            },
            "freeStorage": {
                "title": "Amount of available storage in Mbytes",
                "type": "number",
                "required": true
            },
            "androidState": {
                "description": "Current state in an Android device",
                "type": "object",
                "properties": {
                    "keyboardState": {
                        "title": "Current keyboard state",
                        "type": [ {
                            "enum": [0],
                            description: "Keyboard not hidden"
                            },
                            "enum": [1],
                            description: "Keyboard hidden"
                            },
                            "enum": [2],
                            description: "Undefined"
                            }],
                        "required": true
                    },
                }
            },
            "battery": {
                "title": "Battery level from 0 to 100",
                "type": "number",
                "required": true
            },
            "freeMemory": {
```

```
        "title": "Amount of available memory in Mbytes",
        "type": "number",
        "required": true
    },
    "connectionType": {
        "title": "Current connection type",
        "type": "string",
        "required": true
    },
    "carrier": {
        "title": "Carrier of device",
        "type": "string",
        "required": true
    },
    "networkReachability": {
        "title": "Current network reachability",
        "type": [ {
            "enum": [0],
            description: "Unknown"
            },
            "enum": [1],
            description: "NotReachable"
            },
            "enum": [2],
            description: "ReachableViaWIFI"
            },
            "enum": [3],
            description: "ReachableViaWWAN"
        }],
        "required": true
    },
    "ip": {
        "title": "Ip address of device",
        "type": "string",
        "required": true
    }
},
"additionalProperties" : false
"clientState": {
"description": "Logical web page being loaded for UIC",
"type": "object",
"properties": {
    "pageWidth": {
        "title": "Width of the document of the web page",
        "type": "integer",
        "required": true
    },
    "pageHeight": {
        "title": "Height of the document of the web page",
        "type": "integer",
        "required": true
    },
    "viewPortWidth": {
        "title": "Width of viewport",
        "type": "integer",
        "required": true
    },
    "viewPortHeight": {
        "title": "Height of viewport",
        "type": "integer",
        "required": true
    },
    "viewPortX": {
        "title": "x position of scrollbar on viewport",
        "type": "integer",
        "required": true
    },
            "viewPortY": {
                "title": "y position of scrollbar on viewport",
                "type": "integer",
                "required": true
            },
            "event": {
                "title": "event that triggered the client state",
                "type": "string",
                "required": true
            },
     "deviceScale": {
      "title": "scaling factor for fitting
      page into window for replay",
      "type": "integer",
      "required": true
     },
     "viewTime": {
                "title": "time in milliseconds user was on the event triggered",
                "type": "integer",
                "required": true
            },
            "viewPortXStart": {
                "title": "initial start x position of scrollbar on viewport",
                "type": "integer",
                "required": only used in scroll events
            },
            "viewPortYStart": {
                "title": "initial start y position of scrollbar on viewport",
                "type": "integer",
                "required": only used in scroll events
            },
        },
        "additionalProperties" : false
    }
}
```

Client State (Type 1) message example

This is an example of a Client State (Type 1) message. This example comes from an Android native application.

```
{
    "offset": 667,
    "screenViewOffset": 4556,
    "type": 1,
    "mobileState": {
        "orientation": 0,
        "freeStorage": 33972224,
        "androidState": {
            "keyboardState": 0
        },
        "battery": 50,
        "freeMemory": 64630784,
        "connectionType": "UMTS",
        "carrier": "Android",
        "networkReachability": "ReachableViaWWAN",
        "ip": "0.0.0.0"
    }
}
```

ScreenView (Type 2) messages

ScreenView messages indicate steps in a visitor's experience with your application. These steps can be logical page views in a web application, screen changes in a mobile application, or steps in a business process. ScreenView messages are Type 2 JSON messages.

In Release 8.5 and earlier, these messages were called Application Context messages.

ScreenView (Type 2) message schema

This is the schema for the ScreenView (Type 2) JSON messages.

```
{
    "$ref" : "MessageHeader",
        "screenview/context": {
        "description": "Logical page being loaded or unloaded",
        "type": "object",
        "properties": {
            "type": {
                "title": "Type of ScreenView - LOAD or UNLOAD",
                "type": "string",
                "required": true
            },
            "name": {
                "title": "Name of the logical page",
                "type": "string",
                "required": true
            },
            "url": {
                "title": "Url of the logical page",
                "type": "string",
                "required": true
            },
            "renderTime": {
                "title": "Time it took page to render, only used in LOAD",
                "type": "integer",
                "required": false
            },
            "referrer": {
                "title": "Previous logical page loaded, only used in LOAD",
                "type": "string",
                "required": false
            }
        },
        "additionalProperties" : false,
        "required": false
    }
}
```

ScreenView (Type 2) message example

This is an example of a ScreenView (Type 2) message. This example contains three ScreenView messages, indicating page load and page unload events.

```
{
            "type": 2,
            "offset": 0,
            "screenviewOffset": 0,
            "count": 1,
            "fromWeb": true,
            "screenview": {
                "type": "LOAD",
                "name": "root",
                "url": "/",
                "referrer": ""
            }
        },

        {
            "type": 2,
            "offset": 40824,
            "screenviewOffset": 0,
            "count": 12,
            "fromWeb": true,
            "screenview": {
                "type": "UNLOAD",
                "name": "root",
                "url": "/",
                "referrer": ""
            }
        }
        {
            "type": 2,
            "offset": 2144,
            "screenViewOffset": 0,
            "count": 9,
            "fromWeb": true,
            "screenview": {
                "type": "LOAD",
                "name": "Ford",
                "url": "/example/dynamic/",
                "referrer": "BMW",
            }
        }
```

Connections (Type 3) messages

Connection messages provide information about how requests or responses are managed by the client application. Connections messages are Type 3 JSON messages.

Connections (Type 3) messages schema

This is the schema for Connections (Type 3) JSON messages.

```
{
    "$ref" : "MessageHeader",
    "connection": {
        "description": "Connection in application",
        "type": "object",
        "properties": {
            "statusCode": {
                "title": "Status code of connection",
                "type": "integer",
                "required": true
            },
            "responseDataSize": {
                "title": "Response data size",
                "type": "number",
                "required": true
            },
            "initTime": {
                "title": "Initial time of connection",
                "type": "number",
                "required": true
            },
            "responseTime": {
                "title": "Response time of connection",
                "type": "number",
                "required": true
            },
            "url": {
                "title": "Url of connection",
                "type": "string",
                "required": true
            },
            "loadTime": {
                "title": "Load time from connection",
                "type": "number",
                "required": true
            }
        },
        "additionalProperties" : false
    }
}
```

Connections example

The following example provides information on the status code of the response returned from example.com.

```
{
     "offset": 03829,
        "type": 3,
        "screenViewOffset": 45560,
    "type": 3,
    "connection": {
         "statusCode": 200,
             "responseDataSize": 0272,
         "initTime": 01333669478556,
             "responseTime": 02237,
             "url": "http://google.com",
             "url": "/store/js/tealeaf/
                     TeaLeafTarget.php??width=540&height=960&orientation=0",
             "loadTime": 0
    }
}
```

Control (Type 4) messages

Control messages are used to log user action and behavior. These messages consist of a control identifier and a value that is returned by the identified control. Control messages are Type 4 JSON messages.

The control identifiers are mapped to specific controls for the submitting client framework. The value can be a number, a text string, or structured data.

Control (Type 4) message schema

This is the schema for Control (Type 4) messages.

The X and Y properties are not present in the UI Capture frameworks.

```
{
    "$ref" : "MessageHeader",
      "offset": {
        "title": "Milliseconds offset from offset
                      for when focusIn of text fields occur",
        "type": "integer",
        "required": true
      },
    "target": {
        "description": "Control being logged",
        "type": "object",
        "properties": {
            "position": {
                "description": "Position of control being logged",
                "type": "object",
                "properties": {
                    "x": {
                        "title": "X of the control",
                        "type": "integer",
                        "required": true
                    },
                    "y": {
                        "title": "Y of the control",
                        "type": "integer",
                        "required": true
                    },
                    "height": {
                        "title": "height of control",
                        "type": "integer",
                        "required": true
                    },
                    "width": {
                        "title": "width of control",
                        "type": "integer",
                        "required": true
                    },
                    "relXY": {
                        "title": "relative X & Y ratio that
                                              can be from 0 to 1 with a
                                              default value of 0.5",
                        "type": "string",
                        "required": true for click events
                    },
                },
                "additionalProperties" : false
            }
            "id": {
                "title": "Id/Name/Tag of control",
                "type": "string",
                "required": true
            },
            idType": {
                "title": "To indicate what id is based on id, name or xPath",
                "type": "integer",
                "required": only for UIC due to replay
            },
            "dwell": {
                "title": "Dwell time of control",
                "type": "integer value that is in milliseconds",
                "required": false
            },
             "visitedCount": {
                "title": "Number of times a form control has
                                  been visited to be filled  by user.",
                "type": "integer",
                "required": false
            },
            "isParentLink": {
                "title": "To indicate if control a A type tag",
                "type": "boolean",
                "required": false only in UIC for usability
            },
            "name": {
                "title": "Name of control",
                "type": "string",
                "required": true in UIC
            },
            "type": {
                "title": "Type of control",
                "type": "string",
                "required": true
            },
            "subType": {
                "title": "SubType of control",
                "type": "string",
                "required": true
            },
                    "tlType": {
                "title": "tlType of control that normalizes
                                  the control type for eventing",
                "type": "string",
                "required": true
            },
                    "prevState": {
                "title": "Previous state of control",
                "type": "object",
                "required": true,
                            "properties": {
                    "?": { // Could be any variable name given by developer
                        "title": "Additional data in string format",
                        "type": "string",
                        "required": false
                    }
            },
            "currState": {
                "title": "Current state of control",
                "type": "object",
                "required": true,
                "properties": {
                    "?": { // Could be any variable name given by developer
                        "title": "Additional data in string format",
                        "type": "string",
                        "required": false
                    }
            }
        },
        "additionalProperties" : false
    }
    "event": {
        "description": "Event from control",
        "type": "object",
        "properties": {
                   "tlEvent": {
                "title": "Tealeaf type of event",
                "type": "string",
                "required": true
            },
            "type": {
                "title": "Type of event",
                "type": "string",
                "required": true
            },
            "subType": {
                "title": "Subtype of event",
                "type": "string",
                "required": true
            }
        },
        "additionalProperties" : false
    }
}
```

Control (Type 4) message example

This is an example of a Control Type 4) message.

This control message identifies the new value (MyDataEntry) of a textbox (id=com.tl.uiwidget:id\/editText_c3_1), in which the visitor was dwelling for 3.586 seconds.

```
{
    "target": {
        "position": {
            "y": 38,
            "height": 96,
            "width": 720,
            "x": 0
        },
        "id": "com.tl.uiwidget:id\/editText_c3_1",
        "dwell": 3586,
        "currState": {
            "text": "MyDataEntry"
        },
        "subType": "TextView",
        "type": "EditText",
        "tlType": "textBox",
        "prevState": {
            "text": ""
        }
    },
    "screenViewOffset": 4706,
    "focusInOffset": 23418,
    "offset": 27004,
    "type": 4,
    "event": {
        "type": "OnFocusChange_Out",
        "tlEvent": "textChange"
    }
}
```

Custom Event (Type 5) messages

The Custom Event messages are used to custom log any event from any place in the application. Custom Event messages are Type 5 JSON messages.

Custom Event (Type 5) message schema

This is the schema for the Custom Event (Type 5) messages.

The only required field is the name of the custom event (name value). Application-specific code must be created to process this logged message type.

```
{
    "$ref" : "MessageHeader",
    "customEvent": {
        "description": "Custom event message",
        "type": "object",
        "properties": {
            "name": {
                "title": "Exception name/type",
                "type": "string",
                "required": true
            },
                        "data": "Additional properties given by developer",
                "type": "object",
                "required": truefalse,
                "properties": {
                    "?": { // Could be any variable name given by developer
                        "title": "Additional data in string format",
                        "type": "string",
                        "required": false
                    }
                        },
        },
        "additionalProperties" : false
    }
}
```

Custom Event (Type 5) message example

This is an example of a Custom Event (Type 5) message. This custom event message provides the name of the custom event (MyEvent_1) and several custom properties in the data section.

```
{
     "type": 5,
        "offset": 17981,
    "screenViewOffset": 4556,
    "customEvent": {
        "name": "MyEvent_1",
        "data": {
            "Foo": "Bar",
            "validationError": "Invalid zipcode.",
            "ajaxPerformance": 56734
        }
    }
}
```

Exception (Type 6) messages

The exceptions messages type records the name and description of an exception occurring on the client application. Exception messages are Type 6 JSON messages.

Exception (Type 6) message schema

This is the schema for the Exception (Type 6) messages.

```
{
    "$ref" : "MessageHeader",
    "exception": {
        "description": "Exception description message",
        "type": "object",
        "properties": {
            "description": {
                "title": "Exception Name",
                "type": "string",
                "required": true
            },
            "name": {
                "title": "Exception name/type",
                "type": "string",
                "required": true
            },
                "stackTrace": {
                "title": "Exception stacktrace given by framework",
                "type": "string",
                "required": true
            },
        },
        "additionalProperties" : false
    }
}
```

Exception (Type 6) message example

This is an example of an Exception (Type 6) message. This exception message indicates a divide-by-zero error and includes a stack trace from the client.

```
{
    "offset": 0,
    "screenViewOffset": 4556,
    "type": 6,
    "exception": {
        "description": "divide by zero",
        "name": "class java.lang.ArithmeticException"
      "stackTrace": "java.lang.ArithmeticException: divide by zero\n\tat
com.tl.uic.test.model.JSONTest.testException(JSONTest.java:391)\n\tat
java.lang.reflect.Method.invokeNative(Native Method)\n\tat
java.lang.reflect.Method.invoke(Method.java:507)\n\tat
android.test.InstrumentationTestCase.runMethod(InstrumentationTestCase.java:204
)\n\tat
android.test.InstrumentationTestCase.runTest(InstrumentationTestCase.java:194)\
n\tat
android.test.ActivityInstrumentationTestCase2.runTest(ActivityInstrumentationTe
stCase2.java:186)\n\tat
junit.framework.TestCase.runBare(TestCase.java:127)\n\tat
junit.framework.TestResult$1.protect(TestResult.java:106)\n\tat
junit.framework.TestResult.runProtected(TestResult.java:124)\n\tat
junit.framework.TestResult.run(TestResult.java:109)\n\tat
junit.framework.TestCase.run(TestCase.java:118)\n\tat
android.test.AndroidTestRunner.runTest(AndroidTestRunner.java:169)\n\tat
android.test.AndroidTestRunner.runTest(AndroidTestRunner.java:154)\n\tat
android.test.InstrumentationTestRunner.onStart(InstrumentationTestRunner.java:5
29)\n\tat
android.app.Instrumentation$InstrumentationThread.run(Instrumentation.java:1448
)\n",
    }
}
```

Performance (Type 7) messages

Performance messages show performance data from a browser. Performance messages are Type 7 JSON messages.

Performance (Type 7) message schema

This is the schema for Performance (Type 7) messages.

```
{
    "$ref" : "MessageHeader",
    "performance": {
        "description": "Performance message",
        "type": "object",
        "properties": {
                    },
        "additionalProperties" : false
    }
}
```

Performance (Type 7) message example

This is an example of a Performance (Type 7) message.

```
{
    "type": 7,
    "offset": 9182,
    "screenviewOffset": 9181,
    "count": 3,
    "fromWeb": true,
    "performance": {
        "timing": {
             "redirectEnd": 0,
             "secureConnectionStart": 0,
             "domainLookupStart": 159,
             "domContentLoadedEventStart": 2531,
             "domainLookupEnd": 159,
             "domContentLoadedEventEnd": 2551,
             "fetchStart": 159,
             "connectEnd": 166,
             "responseEnd": 1774,
             "domComplete": 2760,
             "responseStart": 728,
             "requestStart": 166,
             "redirectStart": 0,
             "unloadEventEnd": 0,
             "domInteractive": 2531,
             "connectStart": 165,
             "unloadEventStart": 0,
             "domLoading": 1769,
             "loadEventStart": 2760,
             "navigationStart": 0,
             "loadEventEnd": 2780,
             "renderTime": 986
            },
         "navigation": {
             "type": "NAVIGATE",
                "redirectCount": 0
            }
        }
    }
```

Web Storage (Type 8) messages

Web Storage messages are any objects that contain information about local storage information on the browser. Web Storage messages are Type 8 JSON messages.

Web Storage (Type 8) message schema

This is the schema for the Web Storage (Type 8) messages.

```
"$ref" : "MessageHeader",
webStorage: {
    key : &ldquo;string&rdquo;,
    value: &ldquo;string&rdquo;,
}
```

Web Storage (Type 8) message example

This is an example of a Web Storage (Type 8) message.

```
{
    type: 8,
    offset: 25,
    screenviewOffset: 23,
    count: 2,
    fromWeb: true,
    webStorage: {
        key: "vistCount"
        value: "5"
    }
}
```

Overstat Hover Event (Type 9) messages

Overstat® Hover Event messages are any object containing information about mouse hover and hover-to-click activity. Overstat Hover Event messages are Type 9 JSON messages.

Overstat Hover Event (Type 9) message schema

This is the schema for Overstat Hover Event (Type 9) messages

```
"$ref" : "MessageHeader",
event: {
    xPath: "string",
    hoverDuration: int,
    hoverToClick: boolean,
    gridPosition: {
        x: int,
        y: int
     }
}
```

Overstat Hover Event (Type 9) message example

This is an example of a Overstat Hover Event (Type 9) message.

```
{
    type: 9,
    offset: 25,
    screenviewOffset: 23,
    count: 2,
    fromWeb: true,
    event: {
        xPath: "[\"ii\"]",
        hoverDuration: 5457,
        hoverToClick: false,
        gridPosition: {
            x: 3,
            y: 2
        }
}
```

Layout (Type 10) messages

Layout messages show the current display layout of a native page. Layout messages are Type 10 JSON messages.

Layout (Type 10) message schema

This is the schema for Layout (Type 10) messages.

```
"$ref" : "MessageHeader",
"layoutControl": {
    "description": "Control on application page",
    "type": "object",
    "properties": {
        "position": {
            "description": "Position of control",
            "type": "object",
            "properties": {
                "x": {
                    "title": "X of the control",
                    "type": "integer",
                    "required": true
                },
                "y": {
                    "title": "Y of the control",
                    "type": "integer",
                    "required": true
                },
                "height": {
                    "title": "height of control",
                    "type": "integer",
                    "required": true
                },
                "width": {
                    "title": "width of control",
                    "type": "integer",
                    "required": true
                }
            },
            "additionalProperties" : false
        }
        "id": {
            "title": "Id/Name/Tag of control",
            "type": "string",
            "required": true
        },
        "type": {
            "title": "Type of control",
            "type": "string",
            "required": true
        },
        "subType": {
            "title": "SubType of control",
            "type": "string",
            "required": true
        },
        "tlType": {
            "title": "tlType of control that normalizes the control
type for eventing",
            "type": "string",
            "required": true
        },
        "currState": {
            "title": "Current state of control",
            "type": "object",
            "required": true,
            "properties": {
                "?": { // Could be any variable name given by developer
                    "title": "Additional data in string format",
                    "type": "string",
                    "required": false
                }
            }
        },
        "style" : {
            "title": "Style of the control",
            "type": "object",
            "required": true,
            "properties": {
                "textColor": {
                    "title": "Text color",
                    "type": "string",
                    "required": true
                },
                "textAlphaColor": {
                    "title": "Text alpha color",
                    "type": "string",
                    "required": true
                },
                "textBGColor": {
                    "title": "Text background color",
                    "type": "string",
                    "required": true
                },
                "textBGAlphaColor": {
                    "title": "Text background alpha color",
                    "type": "string",
                    "required": true
                },
                "bgColor": {
                    "title": "Background color",
                    "type": "string",
                    "required": true
                },
                "bgAlphaColor": {
                    "title": "Background alpha color",
                    "type": "string",
                    "required": true
                }
            }
        }
    },
    "additionalProperties" : false
}
```

Layout (Type 10) message example

This is an example of a Layout (Type 10) message.

```
{

    "offset": 27004,

    "screenviewOffset": 4706,

    "count": 16,

    "fromWeb": false,

    "type": 10,
    "layout": {
        "name": "loginPage",
        "controls": [
            {
                "position": {
                    "y": 38,
                    "height": 96,
                    "width": 720,
                    "x": 0
                },
                "id": "com.tl.uiwidget:id\/userNameLabel",
                "type": "UILabel",
                "subType": "UIView",
                "tlType": "label",
                "currState": {
                    "text": "User name*"
                },
                "style": {
                    "textColor": 16777215,
                    "textAlphaColor": 1,
                    "textBGColor": 0,
                    "textBGAlphaColor": 0,
                    "bgColor": 0,
                    "bgAlphaColor": 0
                }
            },
            {...},
            {...}
        ]
    }
}
```

Gesture (Type 11) messages

Gesture messages are used to log user action and behavior. A Gesture message consists of a control identifier and a the value returned by that control. The control identifiers are mapped to specific controls on the client logging platform. The value can be a number, a text string or structured data. Gesture messages are Type 12 JSON messages.

Gesture (Type 11) message schema

This is the schema for Gesture (Type 11) messages.

Tap event schema

This is the schema for tap events:

```
{
    "$ref" : "MessageHeader",
    "event": {
        "description": "Event from control",
        "type": "object",
        "properties": {
            "tlEvent": {
                "title": "Tealeaf type of event",
                "type": "string",
                "required": true
            },
            "type": {
                "title": "Type of event framework reports",
                "type": "string",
                "required": false
            }
        }
    },
    "touches": {
        "description": "Gestures touch objects per finger.",
        "type": "array",
        "required": true
        "items": {
                "description": "Touch objects per finger starting with intial and
ends with last object when finger is lifted from device.",
                "type": "array",
                "required": true,
                "$ref": "Touch"
            }
        }
    }
}
```

Swipe event schema

The swipe event contains only one touch object which will be the initial location with its corresponding direction and velocity. This is the schema for swipe events:

```
{
    "$ref" : "MessageHeader",
    "event": {
        "description": "Event from control",
        "type": "object",
        "properties": {
            "tlEvent": {
                "title": "Tealeaf type of event",
                "type": "string",
                "required": true
            },
            "type": {
                "title": "Type of event framework reports",
                "type": "string",
                "required": false
            }
        }
    },
    "touches": {
        "description": "Gestures touch objects per finger.",
        "type": "array",
        "required": true
        "items": {
                "description": "Touch objects per finger starting with intial
and ends with last object when finger is lifted from device.",
                "type": "array",
                "required": true,
                "$ref": "Touch"
            }
        }
    },
    "direction": {
        "title": "The direction of the swipe which can be up, down. left or
right.",
        "type": "string",
        "required": true
    },
    "velocityX": {
        "title": "The velocity of this measured in pixels per second along the
x axis",
        "type": "float",
        "required": true
    },
    "velocityY": {
        "title": "The velocity of this measured in pixels per second along the
y axis",
        "type": "float",
        "required": false
    }
}
```

Pinch events

The pinch event contains only an initial touch object per finger and the last touch object per finger, with the corresponding direction. This is the schema for pinch events:

```
{
    "$ref" : "MessageHeader",
    "event": {
        "description": "Event from control",
        "type": "object",
        "properties": {
            "tlEvent": {
                "title": "Tealeaf type of event",
                "type": "string",
                "required": true
            },
            "type": {
                "title": "Type of event framework reports",
                "type": "string",
                "required": false
            }
        }
    },
    "touches": {
        "description": "Gestures touch objects per finger.",
        "type": "array",
        "required": true
        "items": {
                "description": "Touch objects per finger starting with intial and
ends with last object when finger is lifted from device.",
                "type": "array",
                "required": true,
                "$ref": "Touch"
            }
        }
    },
    "direction": {
        "title": "Direction of pinch which can be open or close",
        "type": "string",
        "required": true
    }
}
```

Gesture (Type 11) message example

This is an example of a Gesture (Type 11) message.

Tap events

This example is a gesture message for a tap event:

```
{
    "type": 11,
    "offset": 2220,
    "screenviewOffset": 2022,
    "count": 6,
    "fromWeb": false,
    "event": {
        "tlEvent": "tap",
        "type": "ACTION_DOWN"
    },
    "touches": [
        [
            {
                "position": {
                    "y": 388,
                    "x": 0
                },
                "control": {
                    "position": {
                        "height": 20,
                        "width": 250,
                        "relXY": "0.6,0.8"
                    },
                    "id": "com.tl.uic.appDarkHolo:id/textView1",
                    "type": "TextView",
                    "subType": "View",
                    "tlType": "label"
                }
            }
        ]
    ]
}
```

Swipe event example

The swipe event contains only one touch object which will be the initial location with its corresponding direction and velocity. This example is a message for a swipe event:

```
{
    "type": 11,
    "offset": 2220,
    "screenviewOffset": 2022,
    "count": 6,
    "fromWeb": false,
    "event": {
        "tlEvent": "swipe",
        "type": "ACTION_DOWN"
    },
    "touches": [
        [
            {
                "position": {
                    "y": 388,
                    "x": 400
                },
                "control": {
                    "position": {
                        "height": 100,
                        "width": 100,
                        "relXY": "0.4,0.7"
                    },
                    "id": "com.tl.uic.appDarkHolo:id/imageView1",
                    "type": "ImageView",
                    "subType": "View",
                    "tlType": "image"
                }
            }
        ]
    ],
    "direction": "right",
    "velocityX": 23.2,
    "velocityY": 455.14
}
```

Pinch events

The pinch event contains only an initial touch object per finger and the last touch object per finger, with the corresponding direction. This example is a message for a pinch event:

```
{
    "type": 11,
    "offset": 2220,
    "screenviewOffset": 2022,
    "count": 6,
    "fromWeb": false,
    "event": {
        "tlEvent": "pinch",
        "type": "onScale"
    },
    "touches": [
        [
            {
                "position": {
                    "y": 388,
                    "x": 0
                },
                "control": {
                    "position": {
                        "height": 100,
                        "width": 100,
                        "relXY": "0.6,0.8"
                    },
                    "id": "com.tl.uic.appDarkHolo:id/imageView1",
                    "type": "ImageView",
                    "subType": "View",
                    "tlType": "image"
                }
            },
            {
                "position": {
                    "y": 388,
                    "x": 400
                },
                "control": {
                    "position": {
                        "height": 100,
                        "width": 100,
                        "relXY": "0.4,0.7"
                    },
                    "id": "com.tl.uic.appDarkHolo:id/imageView1",
                    "type": "ImageView",
                    "subType": "View",
                    "tlType": "image"
                }
            }
        ],
        [
            {
                "position": {
                    "y": 388,
                    "x": 800
                },
                "control": {
                    "position": {
                        "height": 100,
                        "width": 100,
                        "relXY": "0.6,0.8"
                    },
                    "id": "com.tl.uic.appDarkHolo:id/imageView1",
                    "type": "ImageView",
                    "subType": "View",
                    "tlType": "image"
                }
            },
            {
                "position": {
                    "y": 388,
                    "x": 500
                },
                "control": {
                    "position": {
                        "height": 100,
                        "width": 100,
                        "relXY": "0.4,0.7"
                    },
                    "id": "com.tl.uic.appDarkHolo:id/imageView1",
                    "type": "ImageView",
                    "subType": "View",
                    "tlType": "image"
                }
            }
        ]
    ],
    "direction": "close"
}
```

DOM Capture (Type 12) messages

DOM Capture messages are objects that contain serialized HTML data (DOM snapshot) of the page. DOM Capture Messages are Type 12 JSON messages.

DOM Capture (Type 12) message schema

This is the schema for the DOM Capture (Type 12) messages.

```
"$ref" : "MessageHeader",
"domCapture": {
    "description": "Serialized HTML snapshot of the document.",
    "type": "object",
    "properties": {
        "dcid": {
            "title": "Unique identifier of this DOM snapshot.",
            "type": "string",
            "required": true
        }
        "charset": {
            "title": "Browser reported charset of the document.",
            "type": "string",
            "required": false
        },
        "root": {
            "title": "Serialized HTML of the document.",
            "type": "string",
            "required": false
        },
        "error": {
            "title": "Error message",
            "type": "string",
            "required": false
        },
        "errorCode": {
            "title": "Error code corresponding to the error message.",
            "type": "integer",
            "required": false
        },
        "frames": {
            "title": "Serialized HTML of any child frames of the document",
            "type": "array",
            "required": false,
            "Item": {
                "title": "An object containing serialized HTML of the frame",
                "type": "object",
                "required": false,
                "properties": {

                    "tltid": {

                        "title": "Unique identifier for this frame. Same
tltid is added to the serialized HTML source of the parent."

                        "type": "string",
                        "required": true

                    },

                    "charset": {
                        "title": "Browser reported charset of the document.",
                        "type": "string",
                        "required": true
                    },
                    "root": {
                        "title": "Serialized HTML of the document.",
                        "type": "string",
                        "required": true
                    }
                }
            }
        },
        "canvas" : {
            "title": "Serialized data of the canvas snapshot.",
            "type": "array",
            "required": false,
        }
    },
    "additionalProperties" : false
}
```

DOM Capture (Type 12) message example

This is an example of a DOM Capture (Type 12) message.

This example shows a DOM message without frame or iframe capture:

```
{
    // DOM Capture messages use type 12
    "type": 12,

    // The standard UIC message properties
    "offset": 16821,
    "screenviewOffset": 16817,
    "count": 5,
    "fromWeb": true,

    // The DOM Capture data is namespaced in the domCapture object
    "domCapture": {
        // The "root" contains the serialized HTML of the live DOM
        "root": "<html><body>Hello, World</body></html>",

        // The "charset" contains the value of the document.charset
property returned by the browser
        "charset": "ISO-8859-1",

        // The "dcid" property contains the unique string identifying this DOM
Capture within the page instance.
        "dcid": "dcid-1.1414088027401"
    }
}
```

This example shows a DOM capture message with frame and iframe capture:

```
{
    // DOM Capture messages use type 12
    "type": 12,

    // The standard UIC message properties
    "offset": 16821,
    "screenviewOffset": 16817,
    "count": 5,
    "fromWeb": true,

    // The DOM Capture data is namespaced in the domCapture object
    "domCapture": {
        // The "root" contains the serialized HTML of the live DOM
        "root": "<html><body>Hello, World</body></html>",

        // The "charset" contains the value of the document.charset
property returned by the browser
        "charset": "ISO-8859-1",

        // The "dcid" property contains the unique string identifying this
DOM Capture within the page instance.
        "dcid": "dcid-1.1414088027401"
    }
}
```

This example shows the error message when the captured DOM message length exceeds the configured threshold:

```
{
    // DOM Capture messages use type 12
    "type": 12,

    // The standard UIC message properties
    "offset": 16821,
    "screenviewOffset": 16817,
    "count": 5,
    "fromWeb": true,

    // The DOM Capture data is namespaced in the domCapture object
    "domCapture": {
        // The "error" contains the verbose error message explaining why the
DOM Capture couldn't be performed.
        "error": "Captured length (18045) exceeded limit (10000).",

        // The "errorCode" contains the numeric code for this error message.
Currently, there is only 1 error message.
        "errorCode": 101,

        // The "dcid" property contains the unique string identifying this
DOM Capture within the page instance.
        "dcid": "dcid-1.1414088027401"
    }
}
```

Examples

Below is an example of a message consisting of two sessions that uses all of the message types except the custom event message.

```
{
    "serialNumber": 0,
    "messageVersion": "0.0.0.1",
    "sessions": [
        {
            "startTime": 1328311295574,
            "id": "945202AC4E93104E05EDADE1F6059B97",
            "messages": [
                {
                    "offset": 124,
                    "screenViewOffset": 4556,
                    "type": 2,
                    "logicalPageName": "HomeActivity"
                },
                {
                    "offset": 667,
                    "screenViewOffset": 66778,
                    "type": 1,
                    "mobileState": {
                        "orientation": 0,
                        "freeStorage": 33972224,
                        "androidState": {
                            "keyboardState": 0
                        },
                        "battery": 50,
                        "freeMemory": 64630784,
                        "connectionType": "UMTS",
                        "carrier": "Android",
                        "networkReachability": "ReachableViaWWAN",
                        "ip": "0.0.0.0"
                    }
                },
                {
                    "customEvent": {
                        "name": "Screenshot Taken for file:
                        HomeActivity_1328311296341.jpg"
                    },
                    "offset": 855,
                    "screenViewOffset": 4556,
                    "type": 5
                }
            ]
        }
```

```
      ],
    "clientEnvironment": {
        "mobileEnvironment": {
            "android": {
                "keyboardType": "QWERTY",
                "brand": "generic",
                "fingerPrint": "generic/sdk/generic/
                :2.2/FRF91/43546:eng/test-keys"
            },
            "totalMemory": 63422464,
            "totalStorage": 12288,
            "orientationType": "PORTRAIT",
            "appVersion": "1.0.5",
            "manufacturer": "unknown",
            "userId": "android-build",
            "locale": "English (United States)",
            "deviceModel": "sdk",
            "language": "English"
        },
        "width": 0,
        "height": 0,
        "osVersion": "2.2"
    }
}
```

Upgrading the CX Mobile iOS Logging Framework

When you upgrade the IBM Tealeaf CX Mobile iOS Logging Framework, complete the following general tasks.

Note: Some tasks can vary depending on your development and application environments.

1. Review current requirements. See Requirements.
2. Review the package contents. See Package contents.
3. Verify that your application environment is configured to meet the project requirements. See “Install Tealeaf in an Xcode project” on page 9.
4. Verify that the requirement code changes were applied. See “Modify your application to use Tealaf classes” on page 11.
5. Apply the basic configuration.

   Note: The latest version of the iOS Logging Framework includes new configuration requirements. See Basic configuration.
6. Verify that the appropriate content types are being captured and forwarded by the IBM Tealeaf CX Passive Capture Application. See “Traffic capture configuration on the CX Passive Capture Application” on page 25.

   Note: This step turns on the switch to begin capturing and processing data from the mobile application into IBM Tealeaf. Depending on the volume of data, you may want to use the kill switch. See “Traffic volume management” on page 25.
7. Test your upgraded solution.

Chapter 3. Xamarin MonoTouch iOS applications

If you develop an iOS application with Xamarin, you can use IBM Tealeaf capture and replay technology with the iOS MonoTouch Binding Library.

The Xamarin MonoTouch Binding Library exposes the same APIs as the iOS library, in C# style.

Package contents

The package contains the following software components.

- `TLFMontouchBinding/TLFResources.bundle`. This bundle file contains all of the configuration files needed.
- `TLFMontouchBinding/TLFMonotouchBinding.dll`. This is the library that is designed for use on iOS devices. Use this library for development and testing directly on an iOS device, and include it with your shipping application.

Integrating the IBM Tealeaf MonoTouch Logging Framework with your application

To integrate the IBM Tealeaf MonoTouch Logging Framework with your application, complete these steps.

1. Start the `Xamarin IDE` and open your project.
2. Expand the project in the **Project Navigator**. Right click **References**, then choose **Edit References ...**.
3. Select the correct `TLFMonotouchBinding.dll` and make sure that it is visible in the **Selected references** panel. Click **OK** to close this dialog.
4. Right-click the project and click **Add**, then `Add Existing Folder ...`.
5. Select the correct `TLFResources.bundle` and click **Open**.
6. Click **Include All** to make sure both the folder and files are chosen. Click **OK**.
7. In the **Add File to Folder** dialog, make sure that you select **Copy the file to the directory**, and click **OK**. The `TLFResources.bundle` is now added to your project.
8. Expand the bundle, right-click each of the files, and choose **Properties** to open the **Properties panel**. In the panel, make sure each file's **Copy to output directory** attribute is **Always copy**. By default, it is set to **Do not copy**, which can lead to `Fail to load Bundle` exception errors.

Next, you complete the code changes described in the "Code changes" topic.

Code changes

If your project does not have a customized `MonoTouch.UIKit.UIApplication` class, you must edit the `Main.cs`.

Add `using TLFMonotouchBinding;` in the header.

In the `static void Main()`, add

```
 {
   .........
    TLFApplicationHelper.sharedInstance (); //Initialize the Tealeaf framework
    UIApplication.Main (args, "TLFApplication", "AppDelegate");
    //Use TLFApplication instead of default MonoTouch.UIKit.UIApplication
  }
```

If your project has its own customized `MonoTouch.UIKit.UIApplication` class (for example, named `CustomerUIApplication`), but there is no `SendEvent` and `SendAction` methods in `CustomerUIApplication` class.

Verify that your `Main.cs` looks like the following example. You do not need to change anything.

```
static void Main (string[] args)
{
   .........
      UIApplication.Main (args, "CustomerUIApplication", "AppDelegate");
}
```

You must add `SendEvent` and `SendAction` methods in your `CustomerUIApplication.cs` similarly to the following example.

```
    [Register ("CustomUIApplication")]
    public class CustomUIApplication : MonoTouch.UIKit.UIApplication
    {
        public CustomUIApplication () : base()
        {
        }

        public override void SendEvent (UIEvent uievent){

            TLFApplicationHelper.sharedInstance ().sendEvent(uievent);
            base.SendEvent (uievent);

        }

        public override bool SendAction (Selector action, NSObject target,
        NSObject sender, UIEvent forEvent){

            TLFApplicationHelper.sharedInstance ().sendAction(action, target,
            sender,forEvent);
            return base.SendAction (action, target, sender,forEvent);

        }
    }
```

If your project has its own customized `MonoTouch.UIKit.UIApplication` class (for example, named `CustomerUIApplication`), but there is already `SendAction` and `SendEvent` methods in `CustomerUIApplication` class. If so, you only need to insert `TLFApplicationHelper.sharedInstance ().sendEvent(uievent);` into `SendEvent` and `TLFApplicationHelper.sharedInstance ().sendAction(action, target, sender,forEvent);` into `SendAction` like the example above.

How to resolve method swizzling conflicts in TLFMonotouch

In the IBM Tealeaf iOS logging library, the Objective-C method swizzling technique is used to log user interactions. Xamarin uses the same method to build their SDK. You can resolve the "Method cannot be found" exceptions that occur for some classes after you instrument with the IBM Tealeaf `TLFMonotouch Dll`.

The following table lists the classes with methods that trigger this kind of exception error.

Class	Method
MonoTouch.UIKit.UITableViewSource	RowSelected
MonoTouch.UIKit.UIAlertViewDelegate	Clicked
MonoTouch.Foundation.NSUrlConnectionDelegate	ReceivedResponse, FinishedLoading, FailedWithError
MonoTouch.UIKit.UIWebViewDelegate	ShouldStartLoad, LoadStarted, LoadingFinished, LoadFailed
MonoTouch.UIKit.UIPopoverControllerDelegate	ShouldDismiss, DidDismiss
MonoTouch.UIKit.UISplitViewControllerDelegate	WillShowViewController, WillHideViewController, WillPresentViewController

The solution to these exceptions is to replace the default Xamarin class with the IBM Tealeaf `TLFMonotouchBinding` classes in the following table.

Xamarin class	TLFMonotouchBinding class
MonoTouch.UIKit.UITableViewSource	TLFMonotouchBinding.TLUITableView Source
MonoTouch.UIKit.UIAlertViewDelegate	TLFMonotouchBinding.TLUIAlertView Delegate
MonoTouch.Foundation.NSUrlConnection Delegate	TLFMonotouchBinding.TLNSUrlConnection Delegate
MonoTouch.UIKit.UIWebViewDelegate	TLFMonotouchBinding.TLUIWebView Delegate
MonoTouch.UIKit.UIPopoverController Delegate	TLFMonotouchBinding.TLUIPopover ControllerDelegate
MonoTouch.UIKit.UISplitViewController Delegate	TLFMonotouchBinding.TLUISplitView ControllerDelegate

Chapter 4. Guidelines for tuning CX Mobile iOS Logging Framework

After you get started with the IBM Tealeaf CX Mobile iOS Logging Framework, you make configuration changes that data is collected in a way that is easy to analyze and respectful of your users' privacy. You can also tune the framework so that its work does not interfere with your application's performance.

Session identifiers

The CX Mobile iOS Logging Framework is most powerful as part of a complete IBM Tealeaf system that shows the user activity and device information that is captured by the framework alongside your application's own network activity. The key to organizing all these events is the notion of a session.

A session is a set of related actions that are marked by a common identifier. To join the data from the CX Mobile iOS Logging Framework with your application's own data, both sets must share this common identifier.

The best solution for generating session identifiers is to force the first hit of your mobile native application to be a web hit to the web server. Then, the mobile native application can use the generated session identifier as the identifier for the session on the client. Ideally, this value is the `TLTSID` value that is generated for non-mobile sessions. Other methods of generating session identifiers are not officially currently supported.

For more information about configuring sessioning, see “Sessionization for iOS applications” on page 27.

If you cannot force the first hit to be a web server hit, then the mobile native application must generate a session identifier locally. This session identifier is submitted as a cookie. See Chapter 5, “Reference,” on page 65.

Data collection

To collect data with the CX Mobile iOS Logging Framework, you assign logging levels and identify elements in the user interface. You can also collect extra data and custom events.

Establishing logging levels

Each logging element (a user action, event, or environment data item) has a logging level, 1 - 5. The logging levels are nested. For example, all level 1 elements are included in level 2. Therefore, you assign low logging levels for the most important items.

Tip: The logging level is sometimes referred to in the code as a monitoring level and the terms are interchangeable. The framework can have a monitoring level of 0, meaning that no logging takes place. However, each element must have a logging level 1 - 5.

You assign logging levels in the file TLFLevelsConfiguration.plist, part of TLFResources.bundle. Each element is represented by a numeric Item ID, listed in the reference. See Chapter 5, "Reference," on page 65.

Note: Do not modify the IBM Tealeaf provided property lists. Be sure to not change the structure or the key names in the property list files that are located inside the bundle TLFResources.bundle.

The framework maintains a current logging level. The item LoggingLevel inside TLFConfigurableItems.plist is used the first time your application launches. You can change the current logging level at run time by using a method of the TLFApplicationHelper class:

```
-(void)setCurrentMonitoringLevelType:(kTLFMonitoringLevelType)
       monitoringLevelType;
```

The framework remembers this logging level even when the application goes to the background or exits.

Identifying user interface elements

If your view controllers and buttons inherit from the standard iOS classes, like UIViewController and UIButton, the framework can record their activity. The framework also sends the class names of view controllers and controls, so analysts see these names when they study the application's behavior. The framework sends tag values and puts them in the path names that are used to identify captured data. Therefore, choosing unique tag values for your controls make it much easier to analyze data once it reaches the server.

Collecting extra data

The framework can be configured to log extra data through convenience methods that present data to the server in a standard format. Examples follow.

- Error return values, where you can log NSError values.
- Objective-C exceptions, so you can pass in NSException objects from your application's exception handler.
- GPS location coordinates, so you can log location from your application's location change handler.
- Wireless carrier information.

For more information about how to log these events, see Chapter 5, "Reference," on page 65.

Custom events

If the standard elements tracked by the framework are still not enough, you can create your own custom events.

You can use the TLFCustomEvent class to define your own events.

See Chapter 5, "Reference," on page 65.

Privacy protection

Mobile devices contain a lot of personal information. The framework provides multiple layers of protection for your users' private data.

For information about data masking and blocking, see "Data Privacy in Tealeaf Client Frameworks" in the *IBM Tealeaf Client Framework Data Integration Guide*.

Performance optimization

Different techniques are available for optimizing device and network performance.

Kill switch

The kill switch is a control mechanism that prevents the framework from initializing and having any further effect on your application. To disable the framework:

- Disable a page on your server.
- Force the server to return a status code outside the range of 200-399.

The kill switch is checked each time that the application starts.

For the kill switch to function, you must configure it in the file `TLFConfigurableItems.plist` inside `TLFResources.bundle`.

- `KillSwitchEnabled`:`YES` means that the URL is checked before the rest of the framework initializes; `NO` means that the framework is always initialized.
- `KillSwitchURL`: The URL to check. When the page is reachable, the framework initializes. If the page is not reachable, because of network problems or because you disabled it on your server, the framework does not initialize.
- `KillSwitchTimeout`: How long in seconds to wait for a response from the kill switch before the next try or giving up.
- `KillSwitchTimeInterval`: If there is no response from the page, this tells the framework how many seconds to wait before the next try.
- `KillSwitchMaxNumberOfTries`: How many times to try to get a response from the kill switch page before giving up.

Initialization

You can configure the framework to delay its initialization for a fixed time so that it does not interfere with your application's responsiveness when starting. `DelayTimeOfTLFInitialization` inside `TLFConfigurableItems.plist` tells the framework how long to wait, in seconds, before initializing.

Network performance

You can define when the framework synchronizes data with the server so that you can optimize network performance and make sure that logging data gets sent to the server in a timely way.

The framework monitors the following situations. You can set them up to cause a post of data to the server.

- When the application goes to the background, which gives the framework a chance to send the most recent data to the server. The user's intention may be to quit and not run the application soon. See `DoPostAppGoesToBackground`.
- When the application comes from the background, to check for any data that the framework was unable to send. See `DoPostAppComesFromBackground`.

- When the application is started, to check for any data the framework was unable to send, as well as a report of the app crashing last time it was run. See `DoPostAppIsLaunched`.
- When the screen changes, so data is sent to the server when a screen's worth of interactions were logged. See `DoPostOnScreenChange`.

While the application is running, you can also set up posts:

- At regular time intervals. See `DoPostOnIntervals` and `PostTimeIntervals`.
- When you call a method that tells the framework to post. See `ManualPostEnabled` and the method `requestManualServerPost`. See Chapter 5, “Reference,” on page 65.

You can limit the packet size that is used for posting to the server with `PostMessageMaxBytesSize` if you need to make sure that the framework does not take too much time to transmit.

You can also limit the total network activity per launch of your application. See `MaxNumberOfPostsPerActivation` and `MaxNumberOfBytesPerActivation`.

For more information about how each setting affects posting to the server, see Chapter 5, “Reference,” on page 65.

Intelligent local cache

You can configure the framework to balance its use of local storage, memory, and network bandwidth.

- You can turn off the cache, which makes the framework rely on RAM, but prevents it from writing potentially sensitive data to local storage. See `HasToPersistLocalCache`.
- You can adjust the size of the cache in local storage. See `CachedFileMaxBytesSize` and `{MaxLoggedElementsSize`.
- You can adjust the size of what is cached in memory. See `MemoryWarningMaxMemoryBytesSize` to have the framework respond automatically to low memory warnings.

The framework uses logging levels to manage how much it logs, stores, and posts. These are independent from one another. For example, if it can log at a high level and transmit over WiFi at that same level, you can collect much data when the user has a good connection. However, you could set the caching level lower so that if you lose your network connection, only the most important data gets stored and sent later.

See `CachingLevel`, `PostMessageLevelCellular`, and `PostMessageLevelWiFi`.

For more information about how each property affects the local cache, see Chapter 5, “Reference,” on page 65.

Chapter 5. Reference

This section contains reference information about environmental data and events that are submitted by the CX Mobile iOS Logging Framework from the client. Additionally, you can review logging and framework management information and basic troubleshooting steps.

Required framework and library files

The IBM Tealeaf CX Mobile iOS Logging Framework requires the following frameworks to be linked to your application.

Most applications already include `Foundation.framework` and `UIKit.framework`. `SystemConfiguration.framework` is used for controlling and monitoring the framework's networking.

- `Foundation.framework`
- `UIKit.framework`
- `SystemConfiguration.framework`
- `CoreTelephony.framework`
- `libz.dylib`

Logged elements

Every logged element is part of a session, and every session belongs to an application.

Application data

Application data is consistent across each run of your application on a specific device, except for changes to software version numbers after application upgrades. This data is reported in the `[AppEnv]` section of the request.

The user ID is generated by the framework and is unique for each installation of your application on a device.

- It is consistent each time your application starts.
- A single user with multiple devices receives multiple user IDs.

Table 7. Application data

Name	Short Name	Value
Device Model	deviceModel	The model of the device: `Unknown`, `iPad`, `iPhone`, or `iPod`.
User ID	userId	A unique framework-generated ID for this user.
iOS Version	osVersion	The iOS system version of the device.
Application Name	appName	The name of the application.
Application Version	appVersion	The version of the application. For example, CFBundleVersion.

Table 7. Application data (continued)

Name	Short Name	Value
Tag	tag	All the controls on which you would like to create events must have unique ids. For example, if there is a text field for `Total` of prices of all the items in the cart, and on server you want to create an event for `Total > 300`, you should to assign unique ids to the text filed control. This can be done by setting the tag property of the `UIView`.

Environmental data

The framework automatically handles environmental data that is captured when the framework initializes, typically when your application starts, and at regular time intervals during execution. It also provides support for your application to report environmental data that needs special privacy attention or requires special frameworks.

Environmental data can be distributed among multiple hits.

- Data that is captured when the framework initializes generally appears in one of the first hits of a session.
- Data that is captured at regular time intervals appears along with events. Multiple values can be submitted in a single hit where the number of events is low.
- Data that is captured by your application (location and carrier information) can appear at any time, independently or in hits with the other types of environmental data. Multiple values for a single hit can be submitted if your application makes multiple calls to the framework.

Captured at initialization

These values are captured one time per launch of your application when the framework initializes.

Environment data is collected based on a timer. Environment data related to initialization can be, but is not always, submitted on the first hit of the session. As environment data is passed through the framework, it is prioritized based on its logging level. The order that it is posted to the server and even whether it is posted to the server depends on the following.

- Budgets for in-memory and local storage caches
- Network packet size
- The send level for the type of network available to the application

Note: Data is not posted to the server in the order that it was captured.

Name	Description
pixelDensity	Value that is returned by `[[UIScreen mainScreen] scale]`.
deviceWidth	Value that is returned by `[[UIScreen mainScreen] bounds].size.width`.
deviceHeight	Value that is returned by `[[UIScreen mainScreen] bounds].size.height`.

Name	Description
width	Value that is returned by pixelDensity*deviceWidth.
height	Value that is returned by pixelDensity*deviceHeight.
osVersion	Version of iOS running on the device.
totalStorage	Total storage on the device, free+used.
totalMemory	Total memory of the device, free+used.
manufacturer	Apple Inc. on all iOS devices.
userID	Unique user ID generated by the CX Mobile iOS Logging Framework SDK for current instance of the application.
appVersion	Version of the iOS application.
deviceModel	Type of iPhone, iPad, and so on.
appName	Name of the current application.
orientationType	The orientation of the device (PORTRAIT, LANDSCAPE, FLAT, or UNKNOWN).
locale	Current locale (for example, en).
language	Current language (for example, English).
osType	The type of device used during capture.
tag	All the controls on which you would like to create events must have unique ids. For example, if there is a text field for Total of prices of all the items in the cart, and on server you want to create an event for Total > 300, you should to assign unique ids to the text filed control. This can be done by setting the tag property of the UIView.

Captured during execution

These values are captured at a regular time interval you can set for each logging level with TimeBetweenSnapshots in TLFLevelsConfiguration.plist.

By default, the CX Mobile iOS Logging Framework does not enable battery monitoring, which can drain the battery. Apple suggests enabling battery monitoring only when necessary. When disabled, the following values are reported: BatteryLevel = -100

To monitor the battery's state, your application must enable it through the UIDevice class.

Note: According to Apple, the API used to retrieve the battery level from the device is not always in sync with the value that displays on the device. See http://iphonedevelopertips.com/device/display-battery-state-and-level-of-charge.html. In addition, the value is updated only in 5% increments. See http://www.iphonedevsdk.com/forum/iphone-sdk-development/14301-battery-level.html.

Table 8. Captured during execution

Name	Description
freeMemory	The memory that is remaining.
freeStorage	The storage that is remaining.
battery	The value that is returned by ([UIDevice currentDevice].batteryLevel) * 100.

Table 8. Captured during execution (continued)

Name	Description
carrier	The current network carrier.
networkReachability	The network status (Unknown, NotReachable, ReachableViaWiFi, or ReachableViaWWAN).
ip	The IP address of the device.
orientation	0 if [[UIDevice currentDevice] orientation] returns UIDeviceOrientationPortrait, UIDeviceOrientationFaceDown, or UIDeviceOrientationFaceUp. 90 if UIDeviceOrientationLandscapeRight. 180 if UIDeviceOrientationPortraitUpsideDown. 270 if UIDeviceOrientationLandscapeLeft.

User actions and events

With the CX Mobile iOS Logging Framework, you can track every navigation choice, every touched button, and the contents of every field.

Table views

Table view events are posted when the selected row in the posting table view changes.

Table 9. Table views

Name	Short name	Description
Table View Selection Did Change	selectList:valueChange	Posted when the selected row in the posting table view changes.

Text fields

Text field events are posted when a text field loses focus.

Note: A text field end editing event occurs only when the keyboard is hidden, or the text cursor moves to another text field or text view. If the user interacts with other controls before dismissing the keyboard or editing other text, this event can seem to appear out of sequence.

Table 10. Text fields

Name	Short Name	Description
Text Field Did End Editing	textBox:textChange	Posted when a text field loses focus.

Secure text fields

Secure text field events are posted when a security text field loses focus.

Secure text fields are instances of `UITextField`, where the `secureTextEntry` property of its `UITextInputTraits` is set to `YES`.

Table 11. Secure text fields

Name	Short Name	Description
Secure Text Field Did End Editing	textBox:textChange	Posted when a security text field loses focus

The name-value pairs are the same as for "Text fields" on page 68.

Text views

Text view events are posted when a text view loses focus.

Note: Text views can contain a great deal of information. For large text views, try masking the data by setting the masking level to 1. See Chapter 4, "Guidelines for tuning CX Mobile iOS Logging Framework," on page 61.

Table 12. Text views

Name	Short Name	Description
Text View Did End Editing	textBox:textChange	Posted when a text view loses focus. **Note:** A text field end editing event occurs only when the keyboard is hidden or the text cursor moves to another text field or text view. If the user interacts with other controls before dismissing the keyboard or editing other text, this event can seem to appear out of sequence.

Secure text views

Secure text view events are posted when a secure text view loses focus.

Secure text views are instances of `UITextView`, where the `secureTextEntry` property of its `UITextInputTraits` is set to `YES`.

Table 13. Secure text views

Name	Short Name	Description
Secure Text View Did End Editing	textBox:textChange	Posted when a secure text view loses focus.

Alert views

Alert view events are posted for alerts. There are different types of alert views: show and clicked button.

Show

Table 14. Show alert views

Name	Short Name	Description
Alert View Show In View	AlertViewShowInView	Posted when an alert view appears.

Each alert view show event generates a series of name-value pairs.

Name **Value**

baseClass
The base class name, `UIAlertView`.

title The title of the alert view.

message
The message body of the alert view.

Clicked button

Table 15. Clicked button alert views

Name	Short Name	Description
Alert View Clicked Button	AlertViewClickedButton	Posted when an alert view's button is clicked.

Each alert view clicked button event generates a series of name-value pairs.

Name **Value**

baseClass
The name of the class, usually `UIAlertView`.

title The text that appears in the alert view's title bar.

message
The message that is displayed in the body of the alert view.

buttonTitle
The title of the clicked button.

View controllers

View controller events are posted when a view controller either appears or disappears.

Table 16. View controllers

Name	Short Name	Description
View Controller Did Appear	screenview	Posted after a view controller appears. Posted as JSON message `screenview` and type `LOAD`.
View Controller Did Disappear	screenview	Posted after a view controller disappears. Posted as JSON message `screenview` and type `UNLOAD`.

Synchronous server connections

Synchronous server connection events are posted when a request is either sent or results in an error.

Table 17. Synchronous server connections

Name	Short Name	Description
Send Synchronous Request	connection	Posted when a request is sent.
Send Synchronous Request With Error	connection	Posted when a request results in an error.

Asynchronous server connections

Asynchronous server connection events are posted when a request starts, receives a response, completes successfully, or results in an error.

Requests

Table 18. Asynchronous server requests

Name	Short Name	Description
Connection Init	connection	Posted when a request starts.

Responses

Table 19. Asynchronous server responses

Name	Short Name	Description
Connection Did Receive Response	connection	Posted when a request receives a response.

Successful responses

Table 20. Asynchronous server successful responses

Name	Short Name	Description
Connection Did Finish Loading	connection	Posted when a request successfully completes.

Responses with an error

Table 21. Asynchronous server response errors

Name	Short Name	Description
Connection Did Fail With Error	connection	Posted when a request results in an error.

Unhandled exception

Unhandled exception events are posted when there is an unhandled Objective-C exception.

Table 22. Unhandled exception

Item ID	Name	Short Name	Description
241	Exception	exception	Posted when there is an unhandled Objective-C exception.

Error

An error event is logged by a call to the error logging method.

Table 23. Error

Name	Short Name	Description
Error	exception	An NSError logged by a call to one of the `logNSErrorEvent:` methods.

Network connectivity

A network connectivity event is posted when the network status changes.

Table 24. Network connectivity

Name	Short Name	Description
Network Reachability Changed	networkReachability	Posted when the network status changes. The statuses follow. • `Unknown` • `NotReachable` • `ReachableViaWiFi` • `ReachableViaWWAN`

Crash

A crash event is posted when an abnormal termination is detected.

Table 25. Crash

Name	Short Name	Description
Crash	exception	Posted when the CX Mobile iOS Logging Framework notices (during a subsequent run) that the application did not terminate normally while in the foreground for the session in which this event appears.

Button touch events

Button touch events are posted when a button touch is complete.

Table 26. Button touch events

Name	Short Name	Description
Button Touch Up Inside	button:click	Posted when a button touch is complete.

Configurable items

You edit `TLFConfigurableItems.plist` to change the levels that are set for configurable items.

Note: The logging level that is defined by the configuration value `LoggingLevel` is changed by `setCurrentMonitoringLevelType:` and can be read with `currentMonitoringLevelType`. The terms "logging level" and "monitoring level" are interchangeable.

Table 27. Configurable Items

ItemID	Description	Values
`CachedFileMaxBytesSize`	The maximum size for the local cache. At least 10 times `MaxLoggedElementSize`.	Bytes
`CachingLevel`	The current caching level, applies only when `HasToPersistLocalCache` is `YES`	Integer, `0-5`

Table 27. Configurable Items (continued)

ItemID	Description	Values
CompressPostMessage	When set to YES, HTTP POSTs submitted from the CX Mobile iOS Logging Framework are compressed. **Note:** To enable decompression of compress POSTs, some additional server-side configuration can be necessary. See Chapter 2, "Tealeaf iOS Logging Framework Installation and Implementation," on page 5.	YES/NO
DelayTimeOfTLFInitialization	Delay after app launch before the CX Mobile iOS Logging Framework initializes.	Seconds
DisableAutoInstrumentation	When set to YES, the CX Mobile iOS Logging Framework does not automatically instrument the application elements for logging based on the logging level. Some elements are still captured. See "Disabling auto-instrumentation to include advanced custom instrumentation" on page 83.	YES/NO
DisableTLFFrameworkFlush	When set to YES during the disableTealeafFramework call, the CX Mobile iOS Logging Framework posts cached data to the server. When set to NO (default) during the disableTealeafFramework call, the CX Mobile iOS Logging Framework does not post cached data to the server. For more information about the required calls, see "Enable or disable IBM Tealeaf" on page 77.	YES/NO
DoPostAppComesFromBackground	If YES, the CX Mobile iOS Logging Framework sends data to the server when the application comes from background. **Note:** You cannot enable this setting and ManualPostEnabled together.	YES/NO
DoPostAppGoesToBackground	If YES, the CX Mobile iOS Logging Framework sends data to the server when the application goes to the background. **Note:** You cannot enable this setting and ManualPostEnabled together.	YES/NO
DoPostAppIsLaunched	If YES, the CX Mobile iOS Logging Framework sends data to the server when the application starts. **Note:** You cannot enable this setting and ManualPostEnabled together.	YES/NO

Table 27. Configurable Items (continued)

ItemID	Description	Values
DoPostOnIntervals	If YES, the CX Mobile iOS Logging Framework sends data to the server at regular time intervals that are specified by PostMessageTimeIntervals when the application is in the foreground. **Note:** You cannot enable this setting and ManualPostEnabled together.	YES/NO
DoPostOnScreenChange	If YES, the CX Mobile iOS Logging Framework sends data when the screen changes, subject to ScreenTimeNeededToPost.	YES/NO
DynamicConfigurationEnabled	To use the kill switch, DynamicConfigurationEnabled must be set to YES to enable the kill switch. To use the CX Mobile iOS Logging Framework libraries: • If DynamicConfigurationEnabled==YES, you need to call enableTealeafFramework. • If you do not have the kill switch implemented, you need to have DynamicConfigurationEnabled==NO for libraries to load. See Chapter 2, "Tealeaf iOS Logging Framework Installation and Implementation," on page 5.	YES/NO
FilterMessageTypes	If set to TRUE, only the MessageTypes included in the comma-separated list are sent back to the server. If set to FALSE, all message types are sent back to the server.	TRUE/ FALSE
HasToPersistLocalCache	If YES, data is stored in local storage, instead of in memory.	YES/NO
KillSwitchEnabled	If YES, the CX Mobile iOS Logging Framework checks the kill switch target page before starting; if NO the CX Mobile iOS Logging Framework always starts.	YES/NO
KillSwitchMaxNumberOfTries	The number of times the CX Mobile iOS Logging Framework checks for the kill switch URL before giving up.	Integer
KillSwitchTimeInterval	The time to wait before rechecking the kill switch URL if it is not responding.	Seconds
KillSwitchTimeout	The timeout value for the HTTP request that checks for the kill switch.	Seconds
KillSwitchUrl	Defines the URL to check for the kill switch. The framework requires a successful response to initialize when KillSwitchEnabled is set to YES.	URL

Table 27. Configurable Items (continued)

ItemID	Description	Values
LoggingLevel	The initial logging level the first time the application runs. Depending on the configured logging level, a template of selected application elements is configured to be logged and submitted to IBM Tealeaf for capture. See "Logging templates" on page 78.	integer 0-5
LogViewLayoutOnScreenTransition	When set to YES, on every screenview LOAD, a type 9 message is automatically logged.	YES/NO
ManualPostEnabled	If YES, the CX Mobile iOS Logging Framework sends data to the server only when your application calls requestManualServerPost. **Note:** You cannot enable this setting and DoPostOnIntervals together.	YES/NO
MaxLoggedElementsSize	Size of the cache for logged elements.	Bytes
MaxNumberOfBytesPerActivation	Limits the data sent to the server for each launch of the app.	Bytes
MaxNumberOfPostsPerActivation	Limits the number of posts to the server for each launch of the app.	Bytes
MaxStringsLength	Prevents long strings from taking up storage and bandwidth. **Note:** This value must be set to at least 1.	Number of characters
MemoryWarningMaxMemoryBytesSize	The maximum memory that is allowed for cached logged items after the framework receives a memory warning.	Bytes
PercentOfScreenshotsSize	Percentage of screen capture's original pixel dimensions at which posted screen captures are submitted.	integer 0-100
PostMessageDelayTimeToSendData	The time to wait after your application posts data before the framework posts its own.	Seconds
PostMessageLevelCellular	The logging level of events to be sent to the server over the cellular (3G) network. Set to 0 if you only want logging data that is sent over WiFi.	integer 0-5
PostMessageLevelWiFi	The logging level of events to be sent to the server over WiFi when network performance is good.	integer 0-5
PostMessageMaxBytesSize	The maximum size for a post to the server. **Note:** This value must be set to at least 1024.	Bytes
PostMessageMaxTimeToSendData	Used to calculate network quality. **Note:** This value must be set to at least 1.	Seconds

Table 27. Configurable Items (continued)

ItemID	Description	Values
PostMessageSecondLevel	The logging level of events to be sent to the server when the network performance is poor.	integer 0-5
PostMessageTimeIntervals	How often the CX Mobile iOS Logging Framework sends data to the server when DoPostOnIntervals is set to YES. **Note:** This value must be set to be greater than PostMessageTimeout plus PostMessageDelayTimeToSendData.	Seconds
PostMessageTimeout	The timeout for posts by the CX Mobile iOS Logging Framework to the server. While the framework does not receive a server response within this time frame, the framework keeps trying to send data.	Seconds
PostMessageUrl	The URL for posting data to your server **Note:** To force transport from the client framework to the target page by HTTPS, begin the specified URL with the https:// protocol identifier.	URL
ScreenshotFormat	The file format for screen captures.	PNG or JPG
ScreenTimeNeededToPost	When DoPostOnScreenChange is set to YES, the minimum time a screen must be shown to cause a post.	Seconds
SessionizationCookieName	A cookie name that is defined by the user, which is used to track customer action. It is unique within theCX Mobile iOS Logging Framework environment.	TLTSID
SessionTimeout	The time interval for the amount of time a session is kept in the background before a new session is created. If this session is moved to the foreground after this interval, a new session is created. The default value is 30 minutes.	Minutes
SetGestureDetector	Whether user interface gestures are logged for the app. When set to YES, all supported gestures are logged.	YES/NO
TimeIntervalBetweenSnapshots	The time interval for taking snapshots of environmental data.	Seconds

Dynamic configuration items

At run time, the following items can be configured.

sharedApplication API

See “Session management” on page 87.

Configure PostMessageUrl

You set PostMessageUrl with a URL string.

```
- (BOOL) setPostMessageUrl: (NSString *)value
```

Returns `YES` if PostMessageUrl is successfully set.

Declared in `TLFApplicationHelper.h` .

```
[[TLFApplicationHelper sharedInstance] setPostMessageUrl:(NSString *)];
```

Parameter
Description

value PostMessage URL string.

Configure KillSwitchUrl

You set KillSwitchUrl with a URL string.

```
- (BOOL) setKillSwitchUrl: (NSString *)value
```

Returns `YES` if the KillSwitchUrl is successfully set.

Declared in `TLFApplicationHelper.h`.

```
[ [TLFApplicationHelper sharedInstance] setKillSwitchUrl:
(NSString *)];
```

Parameter
Description

value KillSwitch URL string. Do not include any query parameters.

Enable or disable IBM Tealeaf

At run time, you can enable or disable IBM Tealeaf.

Enable

Declared in TLFApplicationHelper.h.

```
[ [TLFApplicationHelper sharedInstance] enableTealeafFramework];
```

Note: Set PostMessageUrl and KillSwitchUrl before initializing framework. If not, the framework initializes with default settings from the configuration file. Also, no information is logged until the framework is initialized.

Disable

Declared in TLFApplicationHelper.h.

```
[ [TLFApplicationHelper sharedInstance] disableTealeafFramework];
```

To enable the posting of this command to the server, you must set `DisableTLFFrameworkFlush` to `YES`.

Screen capture at run time

To capture a screen at run time, use logPrintScreenEvent in your application per your needs.

Note: To enable capture of screens into IBM Tealeaf, you must configure the CX Passive Capture Application to capture binary POSTs of `png` or `jpg` images. See Chapter 2, “Tealeaf iOS Logging Framework Installation and Implementation,” on page 5.

```
[[TLFCustomEvent sharedInstance] logPrintScreenEvent];
```

If needed, you can take a capture of the current screen at any time. Screen captures are bundled with other CX Mobile iOS Logging Framework data and submitted to IBM Tealeaf for capture and processing.

Note: By default, the CX Mobile iOS Logging Framework sends screens for capture over WiFi connections only. Sending screen captures over networks with less bandwidth can affect application and client framework performance. Use screen capture in a limited capacity initially before you expand capture.

Images are captured as grayscale PNG files or compressed JPG files. Images are captured with a resolution that matches the point resolution of the device.

When a screen is captured, the image is saved locally to be submitted with the next POST to the IBM Tealeaf target page. If caching is disabled, the image is saved in memory.

Note: Screen captures cannot be taken from OpenGL ES views.

Logging templates

Logging levels can be set from 0 to 3. Logging templates can be configured to assist with these settings.

You can configure logging templates for the following network types.

Table 28. Logging templates

Network	Configuration Setting	Default Logging Level
Cellular	PostMessageLevelCellular	1
WiFi	PostMessageLevelWiFi	3

When you configure the base logging level for the CX Mobile iOS Logging Framework, a template of items is preselected for logging.

Logging level legend

To configure the logging level, you set the LoggingLevel value in the TLFLevelsConfiguration.plist file.

Use one of the following values.

- If any control event is set to 0, that event is never logged.
- Based on the device network (Cellular or WiFi) and the UI control values in TLFLevelsConfiguration.plist, events are captured and posted to the IBM Tealeaf Target page.

Custom instrumentation

General

To log custom events, you use sharedInstance in TLFCustomEvent.h.

sharedInstance

Returns the shared instance of TLFCustomEvent. Use this instance to log custom events.

```
+ (TLFCustomEvent *)sharedInstance
```

Returns a shared instance of TLFCustomEvent.

Declared in TLFCustomEvent.h.

Error events

To log errors, you use logNSErrorEvent in TLFCustomEvent.h.

logNSErrorEvent:message:[file:line:level:]

Logs an error as described in an NSError instance.

```
- (void)logNSErrorEvent:(NSError *)error message:(NSString *)message
[file:(const] char *)file line:(unsigned int)line
level:(kTLFMonitoringLevelType)level
```

```
- (void)logNSErrorEvent:(NSError *)error message:(NSString *)message
level:(kTLFMonitoringLevelType)level
```

Parameter
Description

error The NSError returned by the SDK or your own method.

message
A message for your own use.

file The source code file name, usually from the FILE preprocessor macro (optional).

line The source code line number, usually from the LINE preprocessor macro (optional).

level The minimum logging level for this error to be logged.

Declared in TLFCustomEvent.h. kTLFMonitoringLevelType is declared in TLFPublicDefinitions.h.

Exception events

Use this method to log exceptions.

logNSExceptionEvent

Requests that the framework logs an exception trapped by your own exception handler. These methods do not use the Cocoa SDK, which is not exception-safe. Sets the Unhandled flag to false.

This example shows how to call the method:

```
- (BOOL)logNSExceptionEvent:(NSException *)exception;
- (BOOL)logNSExceptionEvent:(NSException *)exception dataDictionary:(NSDictionary*)
dataDictionary;
- (BOOL)logNSExceptionEvent:(NSException *)exception dataDictionary:
(NSDictionary*)dataDictionary isUnhandled:(BOOL)unhandled;
```

Parameter
: **Description**

exception
: The caught `NSexception` instance.

  This value is whether the event was successfully logged. Values are `true` or `false`.

dataDictionary
: This value is additional data about the exception.

unhandled
: Indicates whether the exception was caught by an exception handler.

GPS location events

To avoid making unnecessary location updates, and to protect the privacy of your application's users by ensuring that location is reported only when the application has some other reason to request it, location events are not logged automatically. To log location updates, you use logLocationUpdateEventWithLatitude.

logLocationUpdateEventWithLatitude:longitude:level

This method is meant to be called inside your handler for `locationManager:didUpdateToLocation:fromLocation:`. Your application must include the Core Location framework (`CoreLocation.framework`).

```
#import "CoreLocation/CoreLocation.h"
#import "TLFCustomEvent.h"

...

- (void)locationManager:(CLLocationManager *)manager didUpdateToLocation:
(CLLocation *)
newLocation fromLocation:(CLLocation *)oldLocation {
    CLLocationCoordinate2D c = newLocation.coordinate;
    ...
    [[TLFCustomEvent sharedInstance] logLocationUpdateEventWithLatitude:c.latitude
    longitude:longitude];
}
```

```
- (void)logLocationUpdateEventWithLatitude:(double)latitude longitude:(double)
longitude level:(kTLFMonitoringLevelType)level
```

Parameter
: **Description**

latitude
: The latitude to log.

longitude
: The longitude to log.

level
: The minimum logging level for locations.

Declared in `TLFCustomEvent.h`. `kTLFMonitoringLevelType` is declared in `TLFPublicDefinitions.h`.

Kill Switch events

- (BOOL)setDeviceId:(NSString*)value;

This sets the Device ID.

```
[[TLFApplicationHelper sharedInstance] setDeviceId:@"CustomID"];
```

Parameter
  Description

@param
  The string that represents the new Device ID.

@return
  Indicates whether the Device ID was set.

- (NSString*)getDeviceId;

This returns a string representation of the Device ID.

```
[[TLFApplicationHelper sharedInstance] getDeviceId];
```

Parameter
  Description

@return
  A string representation of the Device ID.

Telephony events

Telephony events are not logged automatically, as the Core Telephony framework (`CoreTelephony.framework`) is required. To log carrier information, you use logCarrierEvent.

logCarrierEvent:isoCountryCode:level

You can log carrier information by linking the Core Telephony framework in your application and then including the code that follows.

```
#import "CoreTelephony/CTTelephonyNetworkInfo.h"
#import "CoreTelephony/CTCarrier.h"

...
    CTTelephonyNetworkInfo *networkInfo = [[CTTelephonyNetworkInfo alloc] init];
    CTCarrier *carrier = [networkInfo subscriberCellularProvider];
    [[TLFCustomEvent sharedInstance] logCarrierEvent:[carrier carrierName]
    isoCountryCode:
    [carrier iosCountryCode]level:kTLFMonitoringLevel1];
    [networkInfo release];
...
```

```
- (void)logCarrierEvent:(NSString *)carrierName country:(NSString *)
isoCountryCode level:(kTLFMonitoringLevelType)level
```

Parameter
  Description

carrierName
  The carrier name.

isoCountryCode
  The country code.

level The minimum logging level for carrier events.

Declared in TLFCustomEvent.h. kTLFMonitoringLevelType is declared in TLFPublicDefinitions.h.

Custom events

You can log a specified event with or without also logging an associated value or dictionary.

logEvent:level

Logs a named event with no additional information.

```
- (void)logEvent:(NSString *)eventName level:
(kTLFMonitoringLevelType)level
```

```
- (void)logEvent:(NSString *)eventName
```

Parameter
Description

eventName
The name of the event. Must not contain equal signs or square brackets.

level The minimum logging level for this event (optional, default is 1).

Declared in TLFCustomEvent.h. kTLFMonitoringLevelType is declared in TLFPublicDefinitions.h.

logEvent:value:level

Logs a named event and a value.

```
- (void)logEvent:(NSString *)eventName value:(NSString *)value
level:(kTLFMonitoringLevelType)level
```

```
- (void)logEvent:(NSString *)eventName value:(NSString *)value
```

Parameter
Description

eventName
The name of the event. Must not contain equal signs or square brackets.

value More information that is associated with the event.

level The minimum logging level for this event logged (optional, default is 1).

Declared in TLFCustomEvent.h. kTLFMonitoringLevelType is declared in TLFPublicDefinitions.h.

logEvent:values:level

Logs a named event and associated dictionary. The dictionary is converted to its JSON representation.

Note: To be convertible to a JSON representation, the values of the dictionary must be NSDictionary, NSArray, NSString, NSNumber or NSNull objects.

```
- (void)logEvent:(NSString *)eventName values:(NSDictionary *)values
- (void)logEvent:(NSString *)eventName values:(NSDictionary *)values
level:(kTLFMonitoringLevelType)level
```

Parameter
Description

eventName
The name of the event. Must not contain equal signs or square brackets.

values More data items that are associated with the event.

level The minimum logging level for this event logged (optional, default is 1).

Declared in `TLFCustomEvent.h`. `kTLFMonitoringLevelType` is declared in `TLFPublicDefinitions.h`.

Disabling auto-instrumentation to include advanced custom instrumentation

By default, the CX Mobile iOS Logging Framework automatically instruments your application by using a template of selected items that are based on the configured logging level.

See "Logging templates" on page 78.

As needed, you can configure the CX Mobile iOS Logging Framework for custom instrumentation. A predefined set of events and objects are instrumented in the application, and the rest can be instrumented through custom methods.

Note: Before you begin, complete the initial configuration tasks that are associated with instrumentation. For more information, see Tealeaf iOS Logging Framework Deployment.

- Optionally, you can disable auto-instrumentation. See "Manual instrumentation."
- In your implementation file, import `TLFCustomEvent.h`.
- You can use any of the available APIs to meet your application requirements. See "Custom instrumentation APIs" on page 86.

Manual instrumentation

You turn off the auto-instrumentation feature in the `TLFConfigurableItems.plist` file. When you do so, no method swizzling occurs, the application state is not monitored, and screen changes or any other events are not automatically tracked.

To disable auto-instrumentation, in the `TLFConfigurableItems.plist` file, which is in the `TLFResources.bundle`, set `DisableAutoInstrumentation` flag to `YES`.

Note: Auto-instrumentation is not recommended because of the large configuration effort, high chance of errors, and possibility of incomplete coverage. If you choose to use auto-instrumentation, you are responsible for implementing theses changes.

Required actions

When you use the iOS SDK with auto-instrumentation turned `OFF`, you must configure a set of actions to occur that auto-instrumentation would otherwise do. The list of required actions follows.

- View Controller changes must be logged by using the API `logAppContext` from the `TLFCustomEvent` class.

- HTTP Connection updates must be logged by using the API `logConnection` from the `TLFCustomEvent` class.

  There are three `logConnection` APIs: one each for initialization, successful response, and failure.
- Button click events must be logged by using API `logClickEvent` from the `TLFCustomEvent`.
- `UITableViewCell` tap events must be logged by using the API `logValueChangeEvent` from the `TLFCustomEvent` class.
- Text change events for `UITextField`, `UITextView`, and `UILabel` must be logged by using the API `logTextChangeEvent` from the `TLFCustomEvent` class.
- To sessionize all `NSURLMutableRequest` objects, you use the API `sessionizeRequest` from the `TLFApplicationHelper` class.
- To track all requests that are made by `UIWebView` from the `UIWebViewDelegate shouldStartLoadWithRequest`, you use the API `isTealeafHybridBridgeRequest` from the `TLFApplicationHelper` class.
- To inject the IBM Tealeaf hybrid bridge into the JavaScript for all web page loads from `UIWebViewDelegate webViewDidFinishLoad`, you use the API `InjectTealeafHybridBridgeOnWebViewDidFinishLoad` from the `TLFApplicationHelper` class.

`TLFCustomEvent` class

Use the following information to manually track various events with the `TLFCustomEvent` class.

- `-(BOOL)logAppContext:(NSString*)logicalPageName applicationContext:(NSString*)applicationContext referrer: (NSString*)referrer`
- `-(BOOL)logEvent:(NSString*)eventName values: (NSDictionary*)values;`
- `-(BOOL)logConnection:(NSURLConnection*)connection error: (NSError*)error`

  Use this API to log failures that occur when a connection is attempted; typically from `NSURLConnectionDelegate didFailWithError` or when `sendSynchronousRequest` returns an error. The first parameter is the connection object, and the second parameter is the error that you received.
- `-(BOOL)logConnection: (NSURLConnection*)connection response: (NSURLResponse*)response responseTimeInMilliseconds:(long long)responseTime;`

  Use this API to log successful connections; typically from `NSURLConnectionDelegate didReceiveResponse` or when `sendSynchronousRequest` returns success. The first parameter is the connection object. The second parameter is the response that you received, and the third is the connection's response type in milliseconds.
- `-(BOOL)logConnection:(NSURLConnection*)connection request: (NSURLRequest*)request;`

  Use this API to log connection initialization; typically before or after a `NSURLConnection initWithRequest` call. The first parameter is connection object, and the second parameter is the request object.
- `-(BOOL)logNSURLSession:(NSObject*)urlSession error:(NSError*)error;`

  Use this API to log failures that occur when a connection is attempted; typically from `NSURLConnectionDelegate didFailWithError` or when `sendSynchronousRequest` returns an error. The first parameter is the connection object, and the second parameter is the error that you received.

- -(BOOL)logNSURLSession:(NSObject*)urlSession response:(NSURLResponse*)response responseTimeInMilliseconds:(long long)responseTime;

  Use this API to log successful connections; typically from NSURLConnectionDelegate didReceiveResponse or when sendSynchronousRequest returns success. The first parameter is the connection object. The second parameter is the response that you received, and the third is the connection's response type in milliseconds.
- -(BOOL)logNSURLSession:(NSObject*)urlSession request:(NSURLRequest*)request;

  Use this API to log connection initialization; typically before or after a call NSURLConnection initWithRequest. The first parameter is connection object, and the second parameter is the request object.
- -(BOOL)logClickEvent:(UIView*)view data:(NSDictionary*)data;

  Use this API to log button click events. Call this from your button click event handlers. The first parameter view is the UIButton object on which the click event happened. The second parameter is optional, and is for future use. You can pass Nil for now.
- -(BOOL)logValueChangeEvent:(UIView*)view data: (NSDictionary*)data;

  Use this API to log UITableViewCell tap events. Call this from your UITableViewDelegate didSelectRowAtIndexPath. The first parameter view is the UITableViewCell object on which the tap event happened The second parameter is optional, and is for future use. You can pass Nil for now.
- -(BOOL)logTextChangeEvent:(UIView*)view data: (NSDictionary*)data;

  Use this API to log text change events for UITextField, UITextView, and UILabel. Call this from your application wherever contents of these three controls changed. If you add the UITextViewTextDidEndEditingNotification observer, you can call it from there. The first parameter view is the object of any of UITextField, UITextView, and UILabel whose text was edited. The second parameter is optional, and is for future use. You can pass Nil for now.
- All APIs are blocking calls. They are all optional and can be called based on your application's design and state machine.
- All the APIs return YES if data is logged, and NO in case of failure. The console debug log shows the reason for failure.

TLFApplicationHelper class

- -(BOOL) sessionizeRequest:(NSMutableURLRequest*)request;

  Use this API so that the IBM Tealeaf iOS SDK can add various Headers and Cookies that can be used to tie all the application session hits together on the server. Call this API as soon as you create the NSMutableURLRequest object, and before you start the HTTP connection. The first parameter is the object of NSMutableURLRequest that the IBM Tealeaf SDK updates.
- -(BOOL) isTealeafHybridBridgeRequest:(NSURLRequest*)request webView:(UIWebView*)webView;

  Start this API from UIWebViewDelegate shouldStartLoadWithRequest. The first parameter is object of NSURLRequest, and the second is object of the current UIWebView. The API determines whether the request is specific to and meant for the IBM Tealeaf iOS SDK from the IBM Tealeaf JavaScript SDK. If it is, the API consumes the data that is sent by the IBM Tealeaf JavaScript SDK. If not, handle the request inside your shouldStartLoadWithRequest. For example, if this API returns YES, ignore the request and return NO from shouldStartLoadWithRequest.

It was not an actual page navigation request from your HTML or JavaScript. If this API returns `NO`, handle the request as it came from your own HTML page or JavaScript.

- `-(BOOL) InjectTealeafHybridBridgeOnWebViewDidFinishLoad: (UIWebView *)webView;`

  Use this API to inject IBM Tealeaf specific JavaScript into your web page. The JavaScript injection helps transfer data from the IBM Tealeaf JavaScript CX UI Capture j2 SDK to the IBM Tealeaf Native iOS SDK. The first parameter is the object of `UIWebView` in which the current web page is loaded. Call it every time a new page is loaded into the `UIWebView`. Place it in `UIWebViewDelegate webViewDidFinishLoad`.

Base instrumentation

The objects and events to populate the following sections are automatically instrumented, even if you enable custom instrumentation.

Environmental Data: This data set is automatically captured during initialization. See "Environmental data" on page 66.

Note: User Actions and Behaviors are not captured when auto-instrumentation is disabled. These events must be manually instrumented. See "Logging level legend" on page 78.

Custom instrumentation APIs

The CX Mobile iOS Logging Framework logs many events automatically, but you can also use it to log errors, exceptions, and custom events.

To log custom events, you can use the `TLFCustomEvent` class. This singleton class offers different methods to log custom events.

For convenience, IBM Tealeaf provides standard events for location tracking and wireless carrier recording.

Example

```
//
// [[TLFCustomEvent sharedInstance] logEvent:@"PurchaseConfirmed"];
//
```

API - log event:

You use the logEvent API to log a simple custom event quickly.

```
- (void)logEvent:(NSString*)eventName;
```

Example

```
[[TLFCustomEvent sharedInstance] logEvent:@"EventName1"];
```

API - log event value:

You use the logEvent API to log a custom event and any value that is related to that event.

```
- (void)logEvent:(NSString*)eventName value:(NSString*)value;
```

Example

```
[[TLFCustomEvent sharedInstance] logEvent:@"EventName2" value:@"EventValue2"];
```

API - log event and dictionary of values:

You use the logEvent API to log a custom event and a dictionary of values that are related to that event.

```
- (void)logEvent:(NSString*)eventName values:(NSDictionary*)values;
```

Example

```
[[TLFCustomEvent sharedInstance] logEvent:@"EventName3" values:dictionary];
```

API - log event and set monitoring level:

You use the logEvent API to log a custom event and set TLFMonitoringLevel.

```
- (void)logEvent:(NSString*)eventName level:(kTLFMonitoringLevelType)level;
```

Example

```
[[TLFCustomEvent sharedInstance] logEvent:@"EventName4" level:2];
```

API - log event, value, and set monitoring level:

You use the logEvent API to log a custom event, any related value, and set a specific TLFMonitoringLevel.

```
- (void)logEvent:(NSString*)eventName value:(NSString*)
value level:(kTLFMonitoringLevelType)level;
```

Example

```
[[TLFCustomEvent sharedInstance] logEvent:@"EventName5"
value:@"EventValue5" level:2 ];
```

API - log event, dictionary of values, and set monitoring level:

You use the logEvent API to log a custom event, a dictionary of values, and set a specific TLFMonitoringLevel.

```
- (void)logEvent:(NSString*)eventName values:(NSDictionary*)values
level:(kTLFMonitoringLevelType)level;
```

Example

```
[[TLFCustomEvent sharedInstance] logEvent:@"EventName6" values:
dictionary level:2];
```

Methods for managing the framework

Session management

Whenever possible, the approach for managing session identifiers is to allow your web server to generate the session identifier for insertion into the `TLTSID` field in the request. This session identifier is consumed and used seamlessly in IBM Tealeaf.

See Chapter 4, “Guidelines for tuning CX Mobile iOS Logging Framework,” on page 61.

If necessary, you can generate a session identifier locally by using the CX Mobile iOS Logging Framework. Information about this method follows; however, use this method only if your web server cannot be configured to generate the session identifier.

See Chapter 4, “Guidelines for tuning CX Mobile iOS Logging Framework,” on page 61.

The CX Mobile iOS Logging Framework starts a session whenever the application starts. It must completely terminate to start a new session and not go to the background.

- To generate a session identifier through the IBM Tealeaf CX Mobile iOS Logging Framework, you implement the `sessionIdGeneration` API of `TLFLibDelegate`.
- To acquire the current session identifier, call the `currentSessionId` method of `TLFApplicationHelper`.

This behavior may not make sense for your application. For example, if you want a new session to begin after every successful purchase, you can use the `startNewTLFSession` method of `TLFApplicationHelper`.

The locally generated session ID is reported as a cookie with all framework posts and is automatically stored.

It is stored in the request that is based on how your IBM Tealeaf environment is configured to manage session identifiers. See "Managing Data Sessionization in Tealeaf CX" in the *IBM Tealeaf CX Installation Manual*.

startNewTLFSession

Tells the IBM Tealeaf CX Mobile iOS Logging Framework to start a new session. This does not affect the behavior of the CX Mobile iOS Logging Framework on the client, but helps you to analyze data that is received by the server. For example, at the end of a purchase, you can count further user interactions as belonging to a separate session, so you can call this method whenever the user completes a purchase.

```
- (void)startNewTLFSession
```

Declared in `TLFApplicationHelper.h`.

currentSessionId

Returns the session ID for the current session. This can be a session ID generated by the CX Mobile iOS Logging Framework, but if you implemented the `sessionIdGeneration` method, then it is the session ID last returned by that method.

```
- (NSString *)currentSessionId
```

Returns the current session ID.

Declared in `TLFApplicationHelper.h`.

Performance optimization

You can improve the performance of your application by choosing logging levels carefully, and by advising the CX Mobile iOS Logging Framework about when to post its data to the server.

requestManualServerPost

Requests the CX Mobile iOS Logging Framework to post to the server as soon as possible. It is a good idea to call this method after you finished your own network transmissions. The device shuts down the WiFi and cell radios when there is no activity. Powering up the radio takes time and battery power, so it is better to transmit in bursts. Refer to the "Tuning for Performance and Responsiveness" chapter of the Apple *iOS Application Programming Guide*.

You must set `ManualPostEnabled` in `TLFConfigurableItems.plist` to `YES` for this method to succeed.

```
- (void)requestManualServerPost
```

Declared in `TLFApplicationHelper.h`.

setCurrentMonitoringLevelType

Sets the current logging level, also known as the monitoring level.

```
- (void)setCurrentMonitoringLevelType:(kTLFMonitoringLevelType)
monitoringLevelType
```

Parameter
Description

monitoringLevelType
The new logging level, 0-5. 0 turns off logging.

Declared in `TLFApplicationHelper.h`.

currentMonitoringLevelType

Returns the current logging level, also known as the monitoring level.

```
- (kTLFMonitoringLevelType)currentMonitoringLevelType
```

Returns the current logging level.

Declared in `TLFApplicationHelper.h`. `kTLFMonitoringLevelType` is declared in `TLFPublicDefinitionsHelper.h`.

Delegate callbacks

You can implement some or all of these methods of `TLFLibDelegate` to help the CX Mobile iOS Logging Framework to work with your application and its server.

The easiest way is to add the `TLFLibDelegate` protocol to your application delegate class, as shown.

```
#import "TLFLibDelegate.h"
...
@interface MyAppDelegate : NSObject <UIApplicationDelegate, TLFLibDelegate> {
...
}
```

sessionIdGeneration

Implemented by your application to provide a session ID to the framework. This does not affect how the framework operates on the client, but is useful when analyzing logged data later. This lets you tie sessions on your server with sessions recorded by the CX Mobile iOS Logging Framework.

```
- (NSString *)sessionIdGeneration
```

Returns the session ID for the framework to record with its logs.

Declared in `TLFPublicDefinitions.h`.

Table 29. Delegate callbacks

Go: Tealeaf iOS Logging Framework Reference Guide	Quick Start: Chapter 2, “Tealeaf iOS Logging Framework Installation and Implementation,” on page 5	Chapter 4, “Guidelines for tuning CX Mobile iOS Logging Framework,” on page 61	Reference	Chapter 6, “Sample code,” on page 91

Chapter 6. Sample code

This section contains sample code for use in implementing the IBM Tealeaf CX Mobile iOS Logging Framework. Modify any samples for use with your web server environment or native application.

Server-side KillSwitch sampling function

When KillSwitch is enabled in the client configuration, the CX Mobile iOS Logging Framework queries the KillSwitch URL to determine whether to enable or disable the CX Mobile iOS Logging Framework for that session.

If the CX Mobile iOS Logging Framework is disabled, then the session is not captured and is excluded from the sampled data.

In the samples below, the KillSwitch URL returns `1` to enable the CX Mobile iOS Logging Framework and `0` to disable the CX Mobile iOS Logging Framework.

Sampling function for ASPX

killswitch.aspx

Sample code for ASPX follows.

```
<%@ Page Language="C#" AutoEventWireup="true"%>
<script runat="server">
  public int Sampler()
  {
    Random rand = new Random();
    int nextRandom = rand.Next(1,100);
    int samplepercent = Convert.ToInt32(ConfigurationManager.AppSettings["rate"]);
    if(nextRandom <= samplepercent){
        return 1;
    }
    else{
        return 0;
    }
  }

</script>
<%
    if (ConfigurationManager.AppSettings["killswitchtype"].Equals("percentagesample")) {
%>
    <%= Sampler() %>
<% } else{ } %>
```

Figure 1. killswitch.aspx

web.config configuration file for ASPX

A sample configuration file for ASPX follows.

```
<?xml version="1.0"?>

<!--
  For more information on how to configure your ASP.NET application,
  please visit http://go.microsoft.com/fwlink/?LinkId=169433
-->
<configuration>

  <appSettings>
    <add key="killswitchtype" value="percentagesample"/>
    <add key="rate" value="50"/>
  </appSettings>
</configuration>
```

Figure 2. web.config

Sampling function for JSP

killswitch.jsp

Sample code for JSP follows.

- If the request does not have parameters, the client framework is always disabled.
- If the id request parameter exists, it is used to check the whitelist.
- If the randomsample parameter exists, the percentage rate from the config.properties file is used to determine how the server responds.

Debug Logs: Generated if the debug property is set to true.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@page import="java.util.Properties"%>
<%@page import="java.util.Date" %>
<%@ page import="java.net.*"%>
<%@ page import="java.io.*" errorPage=""%>
<%
    InputStream stream = application
            .getResourceAsStream("/config.properties");
    Properties props = new Properties();
    props.load(stream);
    Boolean DEBUG = false;

    DEBUG = ("true").equals(props.getProperty("debug"));
    String id = request.getParameter("id");
    String randomsample = request.getParameter("randomsample");
    String killSwitchResponse = "";
    String debugstr = "";

    // white list
    if (id != null && !id.isEmpty()) {
        InputStream whitestream = application.getResourceAsStream(props
                .getProperty("WhiteListFile"));
        BufferedReader input = new BufferedReader(
                new InputStreamReader(whitestream));
        String line = "";
        Boolean match = false;
        while ((line = input.readLine()) != null) {
            line = line.trim();
            if (line.equals(id)) {
                killSwitchResponse = "1";
                match = true;
                break;
            }
        }
        input.close();
        if (!match) {
            killSwitchResponse = "0";
        }
    }

    // If kill switch is by sample rate
    else if (randomsample != null) {
        int rand = (int) (Math.random() * 100);
        int sampleRate = Integer.parseInt(props
                .getProperty("samplerate"));
        if (rand <= sampleRate) {
            killSwitchResponse = "1";
        } else {
            killSwitchResponse = "0";
        }
    } else {
        killSwitchResponse = "0";
    }

    out.print(killSwitchResponse);

    //always give the path from root. This way it almost always works.
    String nameOfTextFile = props.getProperty("logfile");
    PrintWriter pw;

    if (DEBUG) {
        try {
            pw = new PrintWriter(new FileOutputStream(nameOfTextFile,
                    true));
            Date date = new java.util.Date();
            debugstr = date.toString() + "\t";
            if (request.getQueryString() != null) {
                debugstr += request.getQueryString();
            }
            if("0".equals(killSwitchResponse))
                pw.println(debugstr + "\tDisable");
            else
                pw.println(debugstr + "\tEnable");
            //clean up
            pw.close();
        } catch (IOException e) {
            out.println(e.getMessage());
        }
    }
%>
```

web.config configuration file for JSP

A sample configuration file for JSP follows.

```
WhiteListFile=whitelist.txt
samplerate =50
debug=true
logfile=/killswitchlog.txt
```

Figure 3. config.properties

Sampling function for PHP

killswitch.php

Sample code for PHP follows.

```
<?php
    $ini_array = parse_ini_file("config.ini", true);
    //print_r($ini_array);

    // if sample by percent
    if($ini_array['configtype']['killswitchtype'] === 'percentagesample'){
        $sampleRate = intval($ini_array['percentagesample']['rate']);
        killbysamplerate($sampleRate);
    }
    // if sample by whitelist
    else {

    }


    function killbysamplerate($sampleRate){
        $randomnumber = rand(1,100);
        if($randomnumber <= $sampleRate){
            echo '1';
        }
        else {
            echo '0';
        }
    }

    function killbywhitelist($whitelistpath){

    }
?>
```

Figure 4. killswitch.php

web.config configuration file for PHP

A sample configuration file for PHP follows.

```
; This is a sample configuration file
; Comments start with ';', as in php.ini

[configtype]
killswitchtype=percentagesample

[percentagesample]
rate = 50

[whitelist]
x
y
z
```

Figure 5. config.ini

Troubleshooting tools

Console messages

When you set the TLF_DEBUG environment variable to a non-zero value, the CX Mobile iOS Logging Framework creates console messages with NSLog showing when it logs and transmits data.

Note: Support for the TLF_DEBUG environment variable and console messages is available for beta testing, and is not necessarily a part of the final release. The message format is subject to change.

Types of console messages

When you set the TLF_DEBUG environment variable to 1, the CX Mobile iOS Logging Framework creates console messages with NSLog showing when it logs and transmits data. You can set this variable in the Build Arguments panel of your project's Scheme.

Sending messages to the console slows down your application, but shows you:

- User actions and how they are being logged.
- Server posts, both when they are being packaged and when they are finally sent.
- Server responses so you can see that logging framework data is being received by your target page.

Tools for debugging

To help debug problems that are found during testing, you use the TLF_DEBUG environment variable and logger view.

Note: Support for the TLF_DEBUG environment variable and logger view is available for beta testing. They are not necessarily a part of the final release. The message formats are subject to change.

Runtime information

To find runtime information, you can check the current version of the CX Mobile iOS Logging Framework and whether the CX Mobile iOS Logging Framework is initialized at runtime.

frameworkVersion

Returns the version string for the CX Mobile iOS Logging Framework that you are running.

```
- (NSString *)frameworkVersion
```

Returns the framework version string.

Declared in TLFApplicationHelper.h.

isTLFEnabled

Checks if the IBM Tealeaf CX Mobile iOS Logging Framework is enabled.

```
- (BOOL)isTLFEnabled
```

Returns YES if the CX Mobile iOS Logging Framework is enabled, NO otherwise.

Declared in `TLFApplicationHelper.h`.

Crashes

During normal operations, accumulated events are written to a local file on the iOS device. If a power failure occurs while some events are contained in the file, the CX Mobile iOS Logging Framework posts the contents of the file on restart of the application.

- If the local file contains no data, nothing is done on restart.
- If the file is corrupted, an error is logged on restart.

Chapter 7. IBM Tealeaf documentation and help

IBM Tealeaf provides documentation and help for users, developers, and administrators.

Viewing product documentation

All IBM Tealeaf product documentation is available at the following website:

https://tealeaf.support.ibmcloud.com/

Use the information in the following table to view the product documentation for IBM Tealeaf:

Table 30. Getting help

To view...	Do this...
Product documentation	On the IBM Tealeaf portal, go to **?** > **Product Documentation**.
Help for a page on the IBM Tealeaf Portal	On the IBM Tealeaf portal, go to **?** > **Help for This Page**.
Help for IBM Tealeaf CX PCA	On the IBM Tealeaf CX PCA web interface, select **Guide** to access the *IBM Tealeaf CX PCA Manual*.

Available documents for IBM Tealeaf products

Use the following table to view a list of available documents for all IBM Tealeaf products:

Table 31. Available documentation for IBM Tealeaf products

IBM Tealeaf products	Available documents
IBM Tealeaf CX	• *IBM Tealeaf Customer Experience Overview Guide* • *IBM Tealeaf CX Client Framework Data Integration Guide* • *IBM Tealeaf CX Configuration Manual* • *IBM Tealeaf CX Cookie Injector Manual* • *IBM Tealeaf CX Databases Guide* • *IBM Tealeaf CX Event Manager Manual* • *IBM Tealeaf CX Glossary* • *IBM Tealeaf CX Installation Manual* • *IBM Tealeaf CX PCA Manual* • *IBM Tealeaf CX PCA Release Notes*

Table 31. Available documentation for IBM Tealeaf products (continued)

IBM Tealeaf products	Available documents
IBM Tealeaf CX	• *IBM Tealeaf CX RealiTea Viewer Client Side Capture Manual* • *IBM Tealeaf CX RealiTea Viewer User Manual* • *IBM Tealeaf CX Release Notes* • *IBM Tealeaf CX Release Upgrade Manual* • *IBM Tealeaf CX Support Troubleshooting FAQ* • *IBM Tealeaf CX Troubleshooting Guide* • *IBM Tealeaf CX UI Capture j2 Guide* • *IBM Tealeaf CX UI Capture j2 Release Notes*
IBM Tealeaf cxImpact	• *IBM Tealeaf cxImpact Administration Manual* • *IBM Tealeaf cxImpact User Manual* • *IBM Tealeaf cxImpact Reporting Guide*
IBM Tealeaf cxConnect	• *IBM Tealeaf cxConnect for Data Analysis Administration Manual* • *IBM Tealeaf cxConnect for Voice of Customer Administration Manual* • *IBM Tealeaf cxConnect for Web Analytics Administration Manual*
IBM Tealeaf cxOverstat	*IBM Tealeaf cxOverstat User Manual*
IBM Tealeaf cxReveal	• *IBM Tealeaf cxReveal Administration Manual* • *IBM Tealeaf cxReveal API Guide* • *IBM Tealeaf cxReveal User Manual*
IBM Tealeaf cxVerify	*IBM Tealeaf cxVerify Administration Manual*
IBM Tealeaf cxView	*IBM Tealeaf cxView User Manual*
IBM Tealeaf CX Mobile	• *IBM Tealeaf CX Mobile Android Logging Framework Guide* • *IBM Tealeaf Android Logging Framework Release Notes* • *IBM Tealeaf CX Mobile Administration Manual* • *IBM Tealeaf CX Mobile User Manual* • *IBM Tealeaf CX Mobile iOS Logging Framework Guide* • *IBM Tealeaf iOS Logging Framework Release Notes*

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Bay Area Lab
1001 E Hillsdale Boulevard
Foster City, California 94404
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Privacy Policy Considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. A cookie is a piece of data that a web site can send to your browser, which may then be stored on your computer as a tag that identifies your computer. In many cases, no personal information is collected by these cookies. If a Software Offering you are using enables you to collect personal information through cookies and similar technologies, we inform you about the specifics below.

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name, and other personal information for purposes of session management, enhanced user usability, or other usage tracking or functional purposes. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

Various jurisdictions regulate the collection of personal information through cookies and similar technologies. If the configurations deployed for this Software Offering provide you as customer the ability to collect personal information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for providing notice and consent where appropriate.

IBM requires that Clients (1) provide a clear and conspicuous link to Customer's website terms of use (e.g. privacy policy) which includes a link to IBM's and Client's data collection and use practices, (2) notify that cookies and clear gifs/web beacons are being placed on the visitor's computer by IBM on the Client's behalf along with an explanation of the purpose of such technology, and (3) to the extent required by law, obtain consent from website visitors prior to the placement of cookies and clear gifs/web beacons placed by Client or IBM on Client's behalf on website visitor's devices

For more information about the use of various technologies, including cookies, for these purposes, See IBM's Online Privacy Statement at: http://www.ibm.com/privacy/details/us/en section entitled "Cookies, Web Beacons and Other Technologies."

Printed in USA