```
//
//  SBAppDelegate.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface SBAppDelegate : UIResponder <UIApplicationDelegate>

// the main window for this application
@property (strong, nonatomic) UIWindow *window;

// the managedobject context & model used for this application's CoreData store managed by coordinator
@property (readonly, strong, nonatomic) NSManagedObjectContext *managedObjectContext;
@property (readonly, strong, nonatomic) NSManagedObjectModel *managedObjectModel;
@property (readonly, strong, nonatomic) NSPersistentStoreCoordinator *persistentStoreCoordinator;

// defines the current active URL connection objects
@property (nonatomic) NSInteger activeConnectionCount;

// determines whether the application is able to communicate with the API server based on the authentication
    credentials
- (BOOL) canCommunicateViaSecureSocketLayersAndAuthentication;

// the save method for the main coredata context
- (void) saveContext;

// return the application documents directory URL
- (NSURL *) applicationDocumentsDirectory;

// when the managed object context saved
- (void) managedObjectContextDidSave:(NSNotification *) notification;

@end
```

```
//
//  SBAppDelegate.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBAppDelegate.h"
#import "SBUser.h"
#import "SBVerificationManager.h"
#import "SBNavigationBar.h"
#import "SBURLSynchronizeOperation.h"
#import "SBAPIRequest.h"
#import "SBOperationManager.h"
#import "SBReservationManager.h"
#import "SBViewCompatabilityFactory.h"

@interface SBAppDelegate()

// gets called when the authorization credentials change, deletes any user specific items
- (void) verificationManagerDidChangeAuthorizationCredentials:(NSNotification *) notification;

// method that start the synchronization with the API database when needed
- (void) startSynchronization;

@end

@implementation SBAppDelegate

// synthesize core data accessors
@synthesize managedObjectContext = _managedObjectContext;
@synthesize managedObjectModel = _managedObjectModel;
@synthesize persistentStoreCoordinator = _persistentStoreCoordinator;

/*  When the application is ready this method gets called with the provided options such as URL etc */
- (BOOL) application:(UIApplication *) application didFinishLaunchingWithOptions:(NSDictionary *) launchOptions
{
    // create the window frame and set active connections default to 0
    self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
    self.activeConnectionCount = 0;

    // Override point for customization after application launch.
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];

    // register for managed object context changes
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(managedObjectContextDidSave:) name:
        NSManagedObjectContextDidSaveNotification object: nil];

    // get notified when the verification authorization credentials change
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector
        (verificationManagerDidChangeAuthorizationCredentials:) name:
        SBVerificationManagerDidRefreshAuthenticationTokenNotification object: nil];

    // the navigation bar class
    Class navigationClass = nil;
    Class toolBarClass = nil;

    // depending on the ios version the correct navigation bar should be loaded
    if ([SBViewCompatabilityFactory cocoaVersion] == UICocoaVersionIdentifier_iOS_7_0 || [SBViewCompatabilityFactory
        cocoaVersion] == UICocoaVersionIdentifier_iOS_Future) {
        navigationClass = NSClassFromString(@"SBNavigationBar_iOS_7_0");
    } else {
        navigationClass = NSClassFromString(@"SBNavigationBar_iOS_6_0");
        toolBarClass = NSClassFromString(@"SBToolBar");
    }

    // prepare the appearance of back bar button items
    //[[UIBarButtonItem appearance] setBackButtonBackgroundVerticalPositionAdjustment: 50 forBarMetrics:
        UIBarMetricsDefault];

    // returns and activates the initial view controller of the loaded UIStoryboard object, set the windows root
        ViewController
    UINavigationController *initialViewController = [[UINavigationController alloc] initWithNavigationBarClass:
        navigationClass toolbarClass: toolBarClass];

    // make the toolbar not translucent and retrieve the starting view controller
    [[initialViewController toolbar] setTranslucent: NO];
    UIViewController *homeViewController = [[UIStoryboard storyboardWithName: @"PhoneStoryboard" bundle: [NSBundle
        mainBundle]] instantiateViewControllerWithIdentifier: @"InitialViewController"];

    // set the home view controller and the application's root view controller
    [initialViewController setViewControllers: [NSArray arrayWithObject: homeViewController]];
    [self.window setRootViewController: initialViewController];

    // call method to make sure authentication credentials are already being loaded
```

```
    if ([self canCommunicateViaSecureSocketLayersAndAuthentication]) {
        [self startSynchronization];
    }

    return YES;
}

/*  When a managed object context saves its contents we want to merge its data  */
- (void) managedObjectContextDidSave:(NSNotification *)notification
{
    // get the managed object context for the main thread
    NSManagedObjectContext *managedObjectContext = [self managedObjectContext];

    // when the managed object context that posted this notification is not the context of the main thread save the
        data, otherwise an infinite loop is created
    if (![[notification object] isEqual: managedObjectContext]) {

        // call the merge operation on the main managed objectcontext
        [managedObjectContext mergeChangesFromContextDidSaveNotification: notification];
    }
}

/*  Start the synchronization with API database when needed */
- (void) startSynchronization
{
    // when not already synchronizing return
    if ([SBURLSynchronizeOperation needsSynchronization]) {

        // create the sync API request
        SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPISynchronizeDefectsRequestType
            timeout: 120];

        // create and configure the sync operation, then start it
        SBURLSynchronizeOperation *synchronizeOperation = [[SBURLSynchronizeOperation alloc] initWithAPIRequest:
            APIRequest options: nil];
        [[SBOperationManager defaultManager] addOperation: synchronizeOperation];
    }
}

/*  When this method gets called all user specific data should be deleted from the CoreData store, this way the user
    wont have access to objects not linked to the token and username   */
- (void) verificationManagerDidChangeAuthorizationCredentials:(NSNotification *) notification
{
    // new credentials means refreshing when needed
    [self startSynchronization];

    // get the managed object context
    NSManagedObjectContext *objectContext = [self managedObjectContext];

    // create fetch request for all reservation objects
    NSFetchRequest *reservationFetchRequest = [NSFetchRequest fetchRequestWithEntityName: @"Reservation"];
    NSError *fetchError = nil;
    NSArray *reservations = [objectContext executeFetchRequest: reservationFetchRequest error: &fetchError];

    // when the error is set the fetch request could not be executed
    if (fetchError == nil) {

        // delete all objects in the array
        NSEnumerator *reservationEnumerator = [reservations objectEnumerator];
        NSManagedObject *currentObject = nil;

        // enumerate through the array and remove the object from the context
        while (currentObject = [reservationEnumerator nextObject]) {
            [objectContext deleteObject: currentObject];
        }
    }
}

/*  Return whethet the application can communicate with SSL and basic authentication    */
- (BOOL) canCommunicateViaSecureSocketLayersAndAuthentication
{
    // check whether the verification manager has authentication credentials available
    return [[SBVerificationManager defaultVerification] hasValidVerificationCredentials];
}

#pragma mark    --
#pragma mark    UIRemotenotification methods

/*  Gets called when the user has allowed the application to receive remote notifications   */
- (void) application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)
    deviceToken
{
    // get the device token as bytes and create a string to copy the bytes to
    const char *data = [deviceToken bytes];
    NSMutableString* token = [NSMutableString string];

    // process each byte in the data
```

```
    for (int i = 0; i < [deviceToken length]; i++) {
        [token appendFormat:@"%02.2hhX", data[i]];
    }

    // set the current user's device token for sending
    [[NSUserDefaults standardUserDefaults] setObject: token forKey: SBUserDefaultsDeviceToken];
}

/*  Gets called when the user has disallowed the application to receive remote notifications    */
- (void) application:(UIApplication *)application didFailToRegisterForRemoteNotificationsWithError:(NSError *)error
{

}

/*  Gets called when a remote notification has been received and the application is still running   */
- (void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo
{
    NSLog(@"Received remote notification: %@", userInfo);
}


#pragma mark    --
#pragma mark    UIApplication notifications

- (void) applicationWillResignActive:(UIApplication *) application
{
    // Sent when the application is about to move from active to inactive state. This can occur for certain types of
        temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the
        application and it begins the transition to the background state.
    // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should
        use this method to pause the game.
}

- (void) applicationDidEnterBackground:(UIApplication *) application
{
    // Use this method to release shared resources, save user data, invalidate timers, and store enough application
        state information to restore your application to its current state in case it is terminated later.
    // If your application supports background execution, this method is called instead of applicationWillTerminate:
        when the user quits.

    // save user defaults and save CoreData state
    [[NSUserDefaults standardUserDefaults] synchronize];
    [self saveContext];
}

- (void) applicationWillEnterForeground:(UIApplication *) application
{
    // Called as part of the transition from the background to the inactive state; here you can undo many of the
        changes made on entering the background.
}

- (void) applicationDidBecomeActive:(UIApplication *) application
{
    // Restart any tasks that were paused (or not yet started) while the application was inactive. If the
        application was previously in the background, optionally refresh the user interface.
}

- (void) applicationWillTerminate:(UIApplication *) application
{
    // destroy the current reservation being made, this way no null values will be shown between reservations
    [[SBReservationManager defaultReservationManager] didCancelReservation];

    // Saves changes in the application's managed object context before the application terminates.
    [self saveContext];
}

#pragma mark    -
#pragma mark    Core Data stack

/*  Saves the context of the main thread    */
- (void) saveContext
{
    // get the main managed object context for the main thread
    NSError *error = nil;
    NSManagedObjectContext *managedObjectContext = [self managedObjectContext];

    // when the managed object context was successfully fetched or created continue
    if (managedObjectContext != nil) {

        // when there are unsaved changes save the context
        if ([managedObjectContext hasChanges] && ![managedObjectContext save: &error]) {

             // Replace this implementation with code to handle the error appropriately.
             // abort() causes the application to generate a crash log and terminate. You should not use this
                 function in a shipping application, although it may be useful during development.
            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
            abort();
```

```
        }
    }
}

// Returns the managed object context for the application.
// If the context doesn't already exist, it is created and bound to the persistent store coordinator for the
    application.
- (NSManagedObjectContext *) managedObjectContext
{
    // return the managed object context when already created
    if (_managedObjectContext != nil) {
        return _managedObjectContext;
    }

    // get the store coordinator for the context
    NSPersistentStoreCoordinator *coordinator = [self persistentStoreCoordinator];

    // the coordinator must be initialized before creating the context
    if (coordinator != nil) {

        // create the context and set the store coordinator
        _managedObjectContext = [[NSManagedObjectContext alloc] init];
        [_managedObjectContext setPersistentStoreCoordinator:coordinator];
    }

    return _managedObjectContext;
}

// Returns the managed object model for the application.
// If the model doesn't already exist, it is created from the application's model.
- (NSManagedObjectModel *) managedObjectModel
{
    // when already created return it
    if (_managedObjectModel != nil) {
        return _managedObjectModel;
    }

    // create and return the new managed object model
    NSURL *modelURL = [[NSBundle mainBundle] URLForResource: @"BouwCloud" withExtension: @"momd"];
    _managedObjectModel = [[NSManagedObjectModel alloc] initWithContentsOfURL:modelURL];
    return _managedObjectModel;
}

// Returns the persistent store coordinator for the application.
// If the coordinator doesn't already exist, it is created and the application's store added to it.
- (NSPersistentStoreCoordinator *) persistentStoreCoordinator
{
    // when already existing return the coordinator
    if (_persistentStoreCoordinator != nil) {
        return _persistentStoreCoordinator;
    }

    // get the store URL file
    NSURL *storeURL = [[self applicationDocumentsDirectory] URLByAppendingPathComponent: @"BouwCloud.sqlite"];

    // create the persistent store coordinator
    NSError *error = nil;
    _persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel: [self
        managedObjectModel]];

    // add default persistent store to the coordinator
    if (![_persistentStoreCoordinator addPersistentStoreWithType: NSSQLiteStoreType configuration: nil URL: storeURL
        options: nil error: &error])
    {
        /*
         Replace this implementation with code to handle the error appropriately.

         abort() causes the application to generate a crash log and terminate. You should not use this function in a
             shipping application, although it may be useful during development.

         Typical reasons for an error here include:
         * The persistent store is not accessible;
         * The schema for the persistent store is incompatible with current managed object model.
         Check the error message to determine what the actual problem was.


         If the persistent store is not accessible, there is typically something wrong with the file path. Often, a
             file URL is pointing into the application's resources directory instead of a writeable directory.

         If you encounter schema incompatibility errors during development, you can reduce their frequency by:
         * Simply deleting the existing store:
         [[NSFileManager defaultManager] removeItemAtURL:storeURL error:nil]

         * Performing automatic lightweight migration by passing the following dictionary as the options parameter:
         @{NSMigratePersistentStoresAutomaticallyOption:@YES, NSInferMappingModelAutomaticallyOption:@YES}

         Lightweight migration will only work for a limited set of schema changes; consult "Core Data Model
```

```
             Versioning and Data Migration Programming Guide" for details.

         */
        NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
        abort();
    }

    return _persistentStoreCoordinator;
}

#pragma mark - Application's Documents directory

// Returns the URL to the application's Documents directory.
- (NSURL *)applicationDocumentsDirectory
{
    return [[[NSFileManager defaultManager] URLsForDirectory: NSDocumentDirectory inDomains: NSUserDomainMask]
        lastObject];
}

@end
```

```
//
//  main.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBAppDelegate.h"

/*  the main function for the iOS application, the application will start here  */
int main(int argc, char * argv[])
{
    // creates the autorelease pool for memory management and starts the main UIApplication method with the class
       name that we want the applicationdidFinishLaunching method to be called on
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil, NSStringFromClass([SBAppDelegate class]));
    }
}
```

```
/*
 Localizable.strings
 BouwCloud

 Created by Stephan de Bakker on 24-07-13.
 Copyright (c) 2013 Stephan de Bakker. All rights reserved.
 */

/*  Strings for the SBHomeViewController object */
/*  Strings holds the title of the view controller  */
"home.viewcontroller.title" = "REPARATIE APP";

/*  String holds the title of the creating new request cell title   */
"home.viewcontroller.tableview.create.title" = "VERZOEK INDIENEN";

/*  String holds the title of the current request view cell title   */
"home.viewcontroller.tableview.current.title" = "LOPENDE VERZOEKEN";

/*  String holds the title of the my info page cell title   */
"home.viewcontroller.tableview.info.title" = "MIJN GEGEVENS";

/*  String with the description that is shown in the home view controller navigation bar    */
"home.viewcontroller.navigation.description" = "Welkom in de reparatie app, dit is het hoofdscherm.";

/*  Text shown in the middle cell when the reservation is being refreshed   */
"home.viewcontroller.load.description.refresh" = "Status controleren...";

/*  Text shown when a reservation is being uploaded */
"home.viewcontroller.load.description.upload" = "Reparatieverzoek(en) versturen...";

/*  When both refreshing and uploading show this text   */
"home.viewcontroller.load.description.both" = "Versturen & status controleren...";




/*  Strings for the SBPersonalDataViewController object */
/*  String holds the title of the view  */
"data.viewcontroller.title" = "UW GEGEVENS";

/*  The description for this view controller    */
"data.viewcontroller.navigation.description" = "Vul hier je gegevens in die gebruikt worden om een afspraak te
    maken.";

/*  The first section title description in the tableview    */
"data.viewcontroller.tableview.corporation.title" = "WONINGCORPORATIE";

/*  The second section for the personal information like name   etc */
"data.viewcontroller.tableview.personal.title" = "PERSOONLIJKE GEGEVENS";

/*  The third section is for address information    */
"data.viewcontroller.tableview.address.title" = "ADRESGEGEVENS";

/*  The title of the save button in the toolbar  */
"data.viewcontroller.toolbar.save.title" = "GEGEVENS OPSLAAN";

/*  The title of the save button for cancelling */
"data.viewcontroller.toolbar.cancel.title" = "ANNULEREN";

/*  The placeholder for the username textfield  */
"data.viewcontroller.textfield.username.placeholder" = "Naam";

/*  The placeholder for the email textfield */
"data.viewcontroller.textfield.email.placeholder" = "Email";

/*  The placeholder for the telephone number textfield  */
"data.viewcontroller.textfield.telephone.placeholder" = "Telefoonnummer";

/*  The placeholder for the zipcode textfield   */
"data.viewcontroller.textfield.zipcode.placeholder" = "Postcode";

/*  the placeholder string for the house numeber + addition */
"data.viewcontroller.textfield.housenumber.placeholder" = "Huisnummer";

/*  the placeholder for the address or street name textfield    */
"data.viewcontroller.textfield.street.placeholder" = "Adres";

/*  The placeholder for the town name textfield */
"data.viewcontroller.textfield.town.placeholder" = "Woonplaats";

/*  the placeholder for the corporation textfield   */
"data.viewcontroller.textfield.corporation.placeholder" = "Selecteer uw woningcorporatie";

/*  The text displayed in the toolbar keyboard info pane for username   */
"data.viewcontroller.keyboard.username.title" = "Naam";
```

```
/*  The text displayed in the toolbar keyboard info pane for Email   */
"data.viewcontroller.keyboard.email.title" = "Email";

/*  The text displayed in the toolbar keyboard info pane for telephone number   */
"data.viewcontroller.keyboard.telephone.title" = "Telefoonnummer";

/*  The text displayed in the toolbar keyboard info pane for zipcode   */
"data.viewcontroller.keyboard.zipcode.title" = "Postcode";

/*  The text displayed in the toolbar keyboard info pane for house number   */
"data.viewcontroller.keyboard.housenumber.title" = "Huisnummer";



/*  The location selection view controller object   */
/*  The title of the view controller    */
"location.viewcontroller.title" = "VERZOEK INDIENEN";

/*  The description used in the navigation bar  */
"location.viewcontroller.detail.description" = "Kies de locatie waar het defect zich bevindt.";

/*  The detail view title used in the navigation bar    */
"location.viewcontroller.detail.title" = "LOCATIE";

/*  The title of the cell that indicates the outside of the house   */
"location.viewcontroller.outside.cell.title" = "BUITENKANT WONING";

/*  The title of the cell that indicates the inside of the house    */
"location.viewcontroller.inside.cell.title" = "IN DE WONING";

/*  The title of the cell that indicates around the house   */
"location.viewcontroller.around.cell.title" = "RONDOM DE WONING";

/*  The title of the cell that indicates the general rooms  */
"location.viewcontroller.general.cell.title" = "ALGEMENE RUIMTEN";

/*  the footnote used in the outside spaces cell    */
"location.viewcontroller.outside.cell.footnote" = "o.a. Gevel, Dak, Balkon";

/*  the footnote used in the inside spaces cell    */
"location.viewcontroller.inside.cell.footnote" = "o.a. Toilet, Woonkamer, Gang";

/*  the footnote used in the around spaces cell    */
"location.viewcontroller.around.cell.footnote" = "o.a. Garage, Tuin, Plantsoen";

/*  the footnote used in the general spaces cell    */
"location.viewcontroller.general.cell.footnote" = "o.a. Gallerij, Trappenhuis, Lift";



/*  Strings for the room selection view controller  */
/*  The title used in the navigation detail bar */
"room.viewcontroller.navigation.detail.title" = "RUIMTE";

/*  The description for when the inside of a house was chosen   */
"room.viewcontroller.navigation.inside.description" = "Om welke ruimte in uw woning gaat het?";

/*  The description for when the outside of a house was chosen   */
"room.viewcontroller.navigation.outside.description" = "Om welk onderdeel van de buitenkant van de woning gaat
    het?";

/*  The description for when the general locations of a house was chosen   */
"room.viewcontroller.navigation.general.description" = "Om welke algemene ruimte in de woning gaat het?";

/*  The description for when the around of a house was chosen   */
"room.viewcontroller.navigation.around.description" = "Om welke ruimte rondom de woning gaat het?";



/*  Titles and description strings for the element selection view controller    */
/*  The title of the navigation bar detail  */
"element.viewcontroller.navigation.detail.title" = "ONDERDEEL";

/*  The description of the detail navigation    */
"element.viewcontroller.navigation.description" = "Welk onderdeel moet gerepareerd worden?";
```

```
/*  Titles and description texts for the defect selection view controller   */
/*  The title used in the detail view   */
"defect.viewcontroller.navigation.detail.title" = "DEFECT";

/*  The description text for the navigation detail view */
"defect.viewcontroller.navigation.description" = "Om wat voor defect gaat het?";




/*  Titles and escription strings for the optional description and image page   */
/*  The title of the detail navigation bar  */
"optional.viewcontroller.navigation.detail.title" = "OPMERKING TOEVOEGEN";

/*  The description that will be shown in the detail navigation bar */
"optional.viewcontroller.navigation.description" = "Voeg optioneel opmerkingen en een foto toe.";

/*  The title of the button when the user has not inserted any values   */
"optional.viewcontroller.toolbar.skip.title" = "OVERSLAAN";

/*  The title of the button when the user has inserted values */
"optional.viewcontroller.toolbar.next.title" = "VOLGENDE";

/*  the optional string text added to the detail title string   */
"optional.viewcontroller.navigation.detail.title.addition" = "OPTIONEEL";

/*  the optional textview placeholder when the text is empty    */
"optional.viewcontroller.textview.placeholder" = "Schrijf uw opmerking...";

/*  The title of the button which will hide the visible keyboard    */
"optional.viewcontroller.textview.toolbar.hide.title" = "Klaar";

/*  the title for the alert message when no image source could be asked for at the moment   */
"optional.viewcontroller.sources.unavailable.title" = "Invoer niet beschikbaar";

/*  The description for the alert message when no image source is available */
"optional.viewcontroller.sources.unavailable.description" = "Op dit moment is op het apparaat geen camera of foto
    bibliotheek beschikbaar, probeer het later nog eens.";

/*  the title of the dismiss button in the alert view   */
"optional.viewcontroller.sources.unavailable.dismiss.title" = "Verbergen";

/*  The title of the actionsheet which makes the user choose between a new image or existing one    */
"optional.viewcontroller.actionsheet.title" = "Wilt u een nieuwe foto maken of een bestaande foto uitkiezen?";

/*  the title of the actionsheet option for taking a new photo  */
"optional.viewcontroller.actionsheet.new.title" = "Nieuwe foto maken";

/*  The title of the actionsheet option for selecting an existing photo */
"optional.viewcontroller.actionsheet.existing.title" = "Bestaande foto kiezen";

/*  The cancel button for the actionsheet   */
"optional.viewcontroller.actionsheet.cancel.title" = "Annuleren";

/*  the alert view title for when no image could be found in the provided dictionary    */
"optional.viewcontroller.result.unavailable.title" = "Afbeelding niet gevonden";

/*  The alert view description for when the image could not be found    */
"optional.viewcontroller.result.unavailable.description" = "De gekozen of gemaakte afbeelding kon niet worden
    gevonden door de applicatie, probeer het opnieuw.";

/*  When processing of the image failed this title can be shown in the message  */
"optional.viewcontroller.processing.failed.title" = "Verwerken mislukt";




/*  Strings and description for the appointment view controller selector    */
/*  The title of the navigation bar detail view */
"appointment.viewcontroller.navigation.detail.title" = "AFSPRAAK MAKEN";

/*  The description that will be used in the detail navigation bar  */
"appointment.viewcontroller.navigation.detail.description" = "Kies een datum en dagdeel, op deze datum zal het
    defect gerepareerd worden.";

/*  The title of the finish button  */
"appointment.viewcontroller.toolbar.finish.title" = "VERZOEK AFRONDEN";

/*  The default text shown in the date selection cell when no option is selected yet    */
"appointment.viewcontroller.cell.noselection" = "KIES EEN DATUM";

/*  The text shown next to the loading indicator    */
"appointment.viewcontroller.cell.loading.description" = "Data verversen...";

/*  The title that is shown when a loading error occured    */
"appointment.viewcontroller.alert.loading.title" = "Laden mislukt";
```

```
/*  the title for the alertview when no dates are available */
"appointment.viewcontroller.alert.noavailable.title" = "Geen data beschikbaar";

/*  The description when an internal server error occured   */
"appointment.viewcontroller.alert.internal.description" = "Door een server fout is de data niet geladen, wilt u
    opnieuw proberen deze informatie te laden?";

/*  the description when no dates are available for selection   */
"appointment.viewcontroller.alert.noselection.description" = "Op dit moment zijn er geen beschikbare data aanwezig
    om een afspraak op te maken. Probeer het later opnieuw.";

/*  The retry title in the alert view   */
"appointment.viewcontroller.alert.retry.title" = "Opnieuw";

/*  The title for the button which does nothing but hide the alert  */
"appointment.viewcontroller.alert.noaction.title" = "Verberg";




/*  Strings and description texts for the reservation overview view controller  */
/*  The title of the first header in the tableview  */
"overview.viewcontroller.tableview.subject.title" = "REPARATIEONDERWERP";

/*  The title of the second header in the overview tableview    */
"overview.viewcontroller.tableview.remark.title" = "OPMERKING";

/*  the title for the cancel reservation button */
"overview.viewcontroller.toolbar.cancel.title" = "ANNULEREN";

/*  the title for the final confirmation of the reservation */
"overview.viewcontroller.toolbar.confirmation.title" = "VERZOEK INDIENEN";

/*  When the reservation is already sent but the user wants to show it again this will be the title of the view */
"overview.viewcontroller.title" = "VERZOEK DETAIL";

/*  When this is the final step in the process of making the new reservation this title is shown    */
"overview.viewcontroller.final.title" = "VERZOEK AFRONDEN";

/*  the button title for when the user wants to cancel an uploaded reservation  */
"overview.viewcontroller.toolbar.cancel.uploaded.title" = "VERZOEK ANNULEREN";

/*  the title for retrying to upload the selected reservation   */
"overview.viewcontroller.toolbar.retry.upload.title" = "OPNIEUW VERSTUREN";

/*  the title for when the reservation cancelling has failed    */
"overview.viewcontroller.cancel.failed.title" = "Annuleren mislukt";

/*  the description with the cancelling failed title    */
"overview.viewcontroller.cancel.failed.description" = "Het annuleren van het reparatieverzoek is mislukt, probeer
    het later opnieuw.";

/*  the title for the alertview asking the user whether to really cancel the reservation creation   */
"overview.viewcontroller.cancel.new.confirm.title" = "Weet u het zeker?";

/*  The message displayed when the user wants to cancel the creation    */
"overview.viewcontroller.cancel.new.confirm.description" = "Wanneer u doorgaat zal het reparatieverzoek volledig
    worden verwijdert en niet verstuurd worden.";

/*  the button title for cancelling the confirm view    */
"overview.viewcontroller.cancel.new.confirm.cancel.title" = "Nee";

/*  The button title for confiming the cancellation of the reservation  */
"overview.viewcontroller.cancel.new.confirm.approve.title" = "Ja";



/*  For the view with all current requests  */
/*  the title used in the navigation bar description view   */
"all.viewcontroller.navigationbar.detail.description" = "De volgende reparatieverzoeken zijn bij ons bekend.";

/*  the title used in the navigation bar    */
"all.viewcontroller.navigationbar.title" = "LOPENDE VERZOEKEN";

/*  Status description used for when the reservation is being uploaded  */
"all.viewcontroller.status.uploading.description" = "Versturen...";

/*  Status description for when the reservation is successfully uploaded but waiting for approval   */
"all.viewcontroller.status.waiting.description" = "Wachten op goedkeuring";

/*  Status description for when the request is approved */
"all.viewcontroller.status.approved.description" = "Goedgekeurd";

/*  Status description when the reservation failed to be uploaded   */
"all.viewcontroller.status.failed.description" = "Fout tijdens versturen";
```

```
/*  Status description when the frequest is denied  */
"all.viewcontroller.status.denied.description" = "Afgewezen";

/*  When the request is executed the finished description is shown  */
"all.viewcontroller.status.finished.description" = "Uitgevoerd";

/*  This text is shown to the user when no reservations are currently shown in the tableview    */
"all.viewcontroller.no.reservation.description" = "Op dit moment zijn er geen lopende reparatieverzoeken, u kan een
    nieuw reparatieverzoek toevoegen via het hoofdscherm onder \"Verzoek indienen\".";

/*  the title for the detail view when no reservations are available    */
"all.viewcontroller.navigationbar.title.no.reservations" = "GEEN REPARATIEVERZOEKEN";

/*  the title for the detail view when 1 or more reservations are available */
"all.viewcontroller.navigationbar.title.reservations" = "%d REPARATIEVERZOEK(EN)";


/*
 *  Overview controller string declarations
 */
/*  the title of the right barbutton item that shows the now wizrad for reservation */
"overview.controller.navigation.new.title" = "Nieuw verzoek";

/*  the title of the overview controller    */
"overview.controller.title" = "Verzoeken";




/*  The following strings correspond with login view controller of the application
 **  In this view controller a user adds its name, location details
 */

/*  The title of the View Controller where users provide user information   */
"login.controller.title" = "Uw gegevens";

/*  The title of the button users will press to continue to location selection  */
"login.controller.navigation.done.title" = "Volgende";

/*  Placeholder title for the username field    */
"login.controller.textfield.name.placeholder" = "Naam";

/*  Placeholder title for the email field   */
"login.controller.textfield.email.placeholder" = "E-mail";

/*  Placeholder for the first telephone field   */
"login.controller.textfield.phone.placeholder" = "Telefoonnummer 1";

/*  Placeholder for the second optional telephone field */
"login.controller.textfield.phone.optional.placeholder" = "Telefoonnummer 2";

/*  Placeholder for the zipcode field   */
"login.controller.textfield.zipcode.placeholder" = "Postcode";

/*  Plcaeholder for the housenumber textfield   */
"login.controller.textfield.housenumber.placeholder" = "Huisnummer";

/*  Placeholder for the street field    */
"login.controller.textfield.street.placeholder" = "Straat";

/*  Placeholder for the town field  */
"login.controller.textfield.town.placeholder" = "Woonplaats";

/*  Section title for the personal details in the tableview */
"login.controller.tableview.section.personal.title" = "Persoonlijke gegevens";

/*  Section title for the address details in the tableview  */
"login.controller.tableview.section.address.title" = "Adresgegevens";

/*  The description of the message to be shown about the corrupted data */
"login.controller.defect.upload.response.invalid" = "De ontvangen informatie is niet geldig, probeer het later
    opnieuw.";

/*  the title of the cell that lets the user select a customer  */
"login.controller.select.housing.title" = "Kies een woningcorporatie";

/*  the description of the message indicating not all values are correct    */
"login.controller.validation.error.description" = "Niet alle velden zijn juist ingevuld, verbeter de velden met een
    rode rand om verder te kunnen gaan.";

/*  The title of the message shown when validation error occured    */
"login.controller.validation.error.title" = "Validatie fout";

/*  title of the view when uploading data and awaiting the response */
"login.controller.defect.upload.waiting" = "Resultaat afwachten...";
```

```
/*  Title of the view when a defect is uploaded */
"login.controller.defect.upload.progress" = "%d van %d Uploaden...";

/*  Title of the view when the upload went successfully */
"login.controller.defect.upload.success.title" = "Verzoek verstuurd";

/*  Title of the controller when upload errors have occured */
"login.controller.defect.upload.error.title" = "Uploaden mislukt";

/*  title of the controller when the user is being uploaded */
"login.controller.defect.upload.user.title" = "Voorbereiden...";

/*  Title of the preparing defect upload string */
"login.controller.defect.upload.preparing.title" = "Verzoek voorbereiden...";

/*  Title for the accessory view cancel button of the keyboard    */
"login.controller.accessory.keyboard.cancel.title" = "Verberg";

/*  Title for the accessory view next button of the keyboard    */
"login.controller.accessory.keyboard.next.title" = "Volgende";

/*  Title for the accessory view previous button of the keyboard    */
"login.controller.accessory.keyboard.previous.title" = "Vorige";

/*  Title for the accessory view done button of the keyboard    */
"login.controller.accessory.keyboard.done.title" = "Klaar";

/*  Toolbar description for the firstname   */
"login.controller.keyboard.firstname.title" = "Naam";

/*  Toolbar description for the email address   */
"login.controller.keyboard.email.title" = "E-mailadres";

/*  Toolbar description for the first phone number  */
"login.controller.keyboard.phonenumber.title" = "Telefoonnummer 1";

/*  Toolbar description for the optional phone number   */
"login.controller.keyboard.optional.phonenumber.title" = "Telefoonnummer 2";

/*  Toolbar description for the zipcode */
"login.controller.keyboard.zipcode.title" = "Postcode";

/*  Toolbar description for the housenumber & addition  */
"login.controller.keyboard.housenumber.title" = "Huisnummer";




/*  titles and escription string for the customer selection view    */
/*  the loading indicator description   */
"customer.controller.loading.description" = "Woningcorporaties laden...";

/*  the title of the customer view controller   */
"customer.controller.title" = "Kies woningcorporatie";

/*  The title for the done button   */
"customer.controller.navigation.done.title" = "Klaar";

/*  The description of the error view for customers, dit not load etc.  */
"customer.controller.load.error.description" = "Er is een fout opgetreden tijdens het laden van de
    woningcorporaties, probeer het later opnieuw.";



/*  Strings that are used in the ocation selection ViewController */
/*  the title of the cancel button */
"location.controller.navigation.cancel.title" = "Annuleer";

/*  The title of the Location view controller   */
"location.controller.title" = "Locatie";



/*  Strings used in the space controller selector  */
/*  the title of the view controller    */
"space.controller.title" = "Ruimte";



/*  Strings used in the element controller selector  */
/*  the title of the view controller    */
"element.controller.title" = "Onderdeel";
```

```
/*  Strings used in the defect controller selector  */
/*  the title of the view controller    */
"defect.controller.title" = "Defect";




/*  Strings used in the photo controller selector  */
/*  the title of the view controller    */
"image.controller.title" = "Foto";

/*  The title of the button to the next view controller */
"image.controller.navigation.next.title" = "Volgende";

/*  The title for the button that shows the options for taking a picture    */
"image.controller.button.select.photo.title" = "Wijzig foto";

/*  Description for the loading view that waits for the image save  */
"image.controller.image.processing.description" = "Foto verwerken...";

/*  The title for the error message that no sources are available for image picking */
"image.controller.image.sources.unavailable.title" = "Foto's niet beschikbaar";

/*  The description for no sources available in the image selection */
"image.controller.image.sources.unavailable.description" = "Het is op dit moment niet mogelijk om een foto te maken
    of te kiezen, probeer het later opnieuw.";

/*  The cancel button title for skipping the image step */
"actionsheet.button.image.cancel.title" = "Afbeelding overslaan";




/*  Strings used in the description controller selector  */
/*  the title of the view controller    */
"description.controller.title" = "Opmerkingen";

/*  the title of the alertview shown when a choice should be made to add a new proceeding   */
"description.controller.continue.choice.title" = "Nieuw verzoek";

/*  the description provided when the user should make a choice between finish or new   */
"description.controller.continue.choice.description" = "Wilt u nog een reparatieverzoek toevoegen of doorgaan naar
    de volgende stap?";

/*  The button title option to continue to date selection   */
"description.controller.continue.finish.title" = "Volgende";

/*  The button title that allows the user to create a new proceeding    */
"description.controller.continue.new.title" = "Nieuw verzoek";




/*  Strings the are used in the date selections ViewController
 */

/*  the description for the alertview that confirms the upload start    */
"date.controller.alertview.confirm.description" = "Wilt u het reparatieverzoek opsturen?";

/*  The title of the view controller    */
"date.controller.title" = "Selecteer datum";

/*  The title of the button that starts the uploading when possible */
"date.controller.navigation.finish.title" = "Versturen";

/*  The description for the date loading view   */
"date.controller.progress.loading.description" = "Datums laden...";

/*  The tableview section title which holds all available titles    */
"date.controller.tableview.section.title" = "Kies een datum";

/*  The description of the error message when JSON data is corrupted    */
"date.controller.result.error.corrupted" = "De ontvangen informatie is niet geldig, probeer het later opnieuw.";

/*  The error message when no results were found, no employees are available    */
"date.controller.result.error.no.availability" = "Er zijn geen vrije data op dit moment beschikbaar, probeer het
    later opnieuw.";
```

```
/*  Reservation status types description and other strings  */
/*  The status that the reservation is being created and / or is not finished yet   */
"reservation.status.creating.title" = "Onvolledig";

/*  The status that the reservation is being uploaded   */
"reservation.status.uploading.title" = "Bezig met versturen...";

/*  The status the the reservation is uploaded but is waiting for approval  */
"reservation.status.waiting.title" = "Verstuurd";

/*  The status approved reservation */
"reservation.status.approved.title" = "Goedgekeurd";

/*  The stats of the reservation being denied   */
"reservation.status.denied.title" = "Afgewezen";

/*  The status of the reservation failed to upload  */
"reservation.status.failed.title" = "Versturen mislukt";

/*  When a defect description string could not be found show this   */
"reservation.defect.unknown" = "Onbekende status";

/*  When the reservation is successfully executed   */
"reservation.status.finished.title" = "Afgerond";

/*  When the reservation was cancelled by the user  */
"reservation.status.cancelled.title" = "Geannuleerd";

/*  The edit title for cancellation */
"reservation.tableview.cancel.title" = "Annuleren";

/*  The edit title for deletion */
"reservation.tableview.delete.title" = "Verwijderen";




/*  Reservation object string titles and description    */
/*  When no date is available this string will be shown */
"reservation.object.unknown.date.description" = "Geen datum geselecteerd";

/*  when no proceeding is vailable show this text   */
"reservation.object.unknown.proceeding.description" = "Nieuw verzoek";

/*  the text shown when no daypart is yet available */
"reservation.object.unknown.daypart.description" = "Geen tijdstip geselecteerd";

/*  The edit title for cancellation */
"reservation.tableview.cancel.title" = "Annuleer";

/*  The edit title for deletion */
"reservation.tableview.delete.title" = "Verwijder";


/*  Strings for default UIButton titels */
/*  title for the next button   */
"button.title.next" = "Volgende";

/*  Title for the previous button   */
"button.title.previous" = "Vorige";

/*  Title for finishing button  */
"button.title.continue" = "Klaar";







/*  the following strings describe titles and description for action sheets */
/*  The title of the button for taking a new photo  */
"actionsheet.button.image.new.title" = "Nieuwe foto";

/*  the title of the button for selecting a picture from the camera roll    */
"actionsheet.button.image.cameraroll.title" = "Selecteer van filmrol";

/*  the title for cancelling the actionsheet    */
"actionsheet.button.cancel.title" = "Annuleer";
```

```
/*  Error messages not belonging to a specific controller   */
/*  When the operation got cancelled this description is localized  */
"operation.connection.cancel.description" = "De actie is onderbroken door het systeem, probeer het later opnieuw.";



/*  The following strings describe UIAlertView titles and descriptions
 */

/*  Success action executed title in an alertview */
"alertview.title.success" = "Gelukt";

/*  Failed action executed title in an alertview    */
"alertview.title.error" = "Foutmelding";

/*  the confirm title for an alert view */
"alertview.title.confirm" = "Reparatie verzoek";

/*  The title for an error with writing an image to disk    */
"alertview.title.save.failed" = "Opslaan mislukt";

/*  the button title for the action that dismisses the alert view   */
"alertview.button.title.dismiss" = "Verberg";

/*  The button title for the the action that lets the user retry */
"alertview.button.title.retry" = "Opnieuw";

/*  The button that lets the user go to a page that should be edited    */
"alertview.button.title.edit" = "Bewerken";

/*  The button that cancels the action that can be made with the alert  */
"alertview.button.title.cancel" = "Annuleer";

/*  the button that confirms an action that will be made    */
"alertview.button.title.confirm" = "Bevestig";




/*  Custom error description strings    */
/*  The description for when the operation has to execute an invalid connection, which happened because of the
    URLRequest being invalid    */
"error.operation.request.invalid.description" = "Het verzoek is mislukt omdat  de URL en content niet is
    goedgekeurd, probeer het opnieuw.";

/*  The description for a cancelled operation   */
"error.operation.cancel.description" = "Het verzoek is geannuleerd, probeer het opnieuw.";

/*  When the received response is invalid this message is used  */
"error.response.invalid.response.description" = "De verkregen informatie is ongeldig, probeer het later opnieuw.";

/*  The image that should be parsed is invalid and could not be created */
"error.response.image.corrupt.description" = "De geladen foto kon niet worden verwerkt door een ongeldig formaat.";

/*  When the application failed to fetch authorization credentials  */
"operation.authorization.request.failed.description" = "Het is niet gelukt om contact te maken met de server, dit is
    benodigd om de applicatie te laten werken. Druk op opnieuw om het nogmaals te proberen.";

/*  Title for authorization failed fetchinh */
"operation.authorization.request.failed.title" = "Authorizatie mislukt";

/*  Title for when cancelling of a reservation has failed   */
"operation.cancel.reservation.failed.title" = "Annuleren mislukt";

/*  Description for when a reservation could not be cancelled   */
"operation.cancel.reservation.failed.description" = "Het verzoek is niet geannuleerd door een fout, probeer het
    later opnieuw.";

/*  The title for twhen a reservation has failed to be uplaoded */
"operation.reservation.upload.failed.title" = "Versturen mislukt";

/*  The description for when a reservation has failed to be uploaded    */
"operation.reservation.upload.failed.description" = "Het versturen van het reparatieverzoek is mislukt, probeer het
    later opnieuw.";

/*  the description for when the image data could not be created    */
"operation.image.process.failed.description" = "Het verwerken van de foto is mislukt, probeer opnieuw een foto toe
    te voegen.";
```

```
/*
  Localizable.strings
  BouwCloud

  Created by Stephan de Bakker on 24-07-13.
  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
*/
```

```
/*
 *  Overview controller string declarations
*/
/*  the title of the right barbutton item that shows the now wizrad for reservation */
"overview.controller.navigation.new.title" = "Report";

/*  the title of the overview controller    */
"overview.controller.title" = "Reports";


/*  The following strings correspond with login view controller of the application
**  In this view controller a user adds its name, location details
*/

/*  The title of the View Controller where users provide user information   */
"login.controller.title" = "Personal Details";

/*  The title of the button users will press to continue to location selection  */
"login.controller.navigation.done.title" = "Next";

/*  Placeholder title for the username field    */
"login.controller.textfield.name.placeholder" = "Your name (required)";

/*  Placeholder title for the email field   */
"login.controller.textfield.email.placeholder" = "email (required)";

/*  Placeholder for the first telephone field   */
"login.controller.textfield.phone.placeholder" = "Phone number (required)";

/*  Placeholder for the second optional telephone field */
"login.controller.textfield.phone.optional.placeholder" = "Phone number (optional)";

/*  Placeholder for the zipcode field   */
"login.controller.textfield.zipcode.placeholder" = "Zipcode";

/*  Plcaeholder for the housenumber textfield   */
"login.controller.textfield.housenumber.placeholder" = "Number";

/*  Placeholder for the street field    */
"login.controller.textfield.street.placeholder" = "Street";

/*  Placeholder for the town field  */
"login.controller.textfield.town.placeholder" = "Town";

/*  Section title for the personal details in the tableview */
"login.controller.tableview.section.personal.title" = "Personal Details";

/*  Section title for the address details in the tableview  */
"login.controller.tableview.section.address.title" = "Address of Defect";

/*  The description that is shown when the defect is successfully uploaded  */
"login.controller.defect.upload.success.description" = "The defect is sent, you will receive an email and
    notification with details about the appointment.";

/*  The description of the message to be shown about the corrupted data */
```

```
"login.controller.defect.upload.response.invalid" = "The received server response is invalid, please try again.";

/*  the title of the cell that lets the user select a customer  */
"login.controller.select.housing.title" = "Select Housing";

/*  the description of the message indicating not all values are correct    */
"login.controller.validation.error.description" = "Not all values could be validated successfully, please correct
    the indicated fields.";

/*  title of the view when uploading data and awaiting the response */
"login.controller.defect.upload.waiting" = "Awaiting response...";

/*  Title of the view when a defect is uploaded */
"login.controller.defect.upload.progress" = "Uploading %d of %d...";

/*  Title of the view when the upload went successfully */
"login.controller.defect.upload.success.title" = "Uploaded Successfully";

/*  Title of the controller when upload errors have occured */
"login.controller.defect.upload.error.title" = "Upload Failed";

/*  title of the controller when the user is being uploaded */
"login.controller.defect.upload.user.title" = "Preparing...";

/*  Title of the preparing defect upload string */
"login.controller.defect.upload.preparing.title" = "Preparing defect...";

/*  Title for the accessory view cancel button of the keyboard    */
"login.controller.accessory.keyboard.cancel.title" = "Hide";

/*  Title for the accessory view next button of the keyboard    */
"login.controller.accessory.keyboard.next.title" = "Next";

/*  Title for the accessory view previous button of the keyboard    */
"login.controller.accessory.keyboard.previous.title" = "Previous";

/*  Title for the accessory view done button of the keyboard    */
"login.controller.accessory.keyboard.done.title" = "Done";




/*  titles and escription string for the customer selection view    */
/*  the loading indicator description   */
"customer.controller.loading.description" = "Loading Customers...";

/*  the title of the customer view controller   */
"customer.controller.title" = "Select Housing";

/*  The title for the done button   */
"customer.controller.navigation.done.title" = "Done";

/*  The description of the error view for customers, dit not load etc.  */
"customer.controller.load.error.description" = "An error occured during loading of the customers";



/*  Strings that are used in the ocation selection ViewController */
/*  the title of the cancel button */
"location.controller.navigation.cancel.title" = "Cancel";

/*  The title of the Location view controller   */
"location.controller.title" = "Location";



/*  Strings used in the space controller selector  */
/*  the title of the view controller    */
"space.controller.title" = "Space";



/*  Strings used in the element controller selector  */
/*  the title of the view controller    */
"element.controller.title" = "Element";



/*  Strings used in the defect controller selector  */
/*  the title of the view controller    */
"defect.controller.title" = "Defect";



/*  Reservation object string titles and description    */
```

```
/*  When no date is available this string will be shown */
"reservation.object.unknown.date.description" = "No date selected";

/*  when no proceeding is vailable show this text   */
"reservation.object.unknown.proceeding.description" = "New reservation";

/*  the text shown when no daypart is yet available */
"reservation.object.unknown.daypart.description" = "No time selected";




/*  Strings used in the photo controller selector  */
/*  the title of the view controller    */
"image.controller.title" = "Photo";

/*  The title of the button to the next view controller */
"image.controller.navigation.next.title" = "Next";

/*  The title for the button that shows the options for taking a picture    */
"image.controller.button.select.photo.title" = "Select a Photo";

/*  Description for the loading view that waits for the image save  */
"image.controller.image.processing.description" = "Processing Image...";

/*  The title for the error message that no sources are available for image picking */
"image.controller.image.sources.unavailable.title" = "Image Unavailable";

/*  The description for no sources available in the image selection */
"image.controller.image.sources.unavailable.description" = "No image sources are available at the time, please try
    again later.";

/*  The cancel button title for skipping the image step */
"actionsheet.button.image.cancel.title" = "Skip image";




/*  Strings used in the description controller selector  */
/*  the title of the view controller    */
"description.controller.title" = "Description";

/*  the title of the alertview shown when a choice should be made to add a new proceeding   */
"description.controller.continue.choice.title" = "Make a choice";

/*  the description provided when the user should make a choice between finish or new   */
"description.controller.continue.choice.description" = "Would you like to create a new defect or continue to date
    selection?";

/*  The button title option to continue to date selection   */
"description.controller.continue.finish.title" = "Select date";

/*  The button title that allows the user to create a new proceeding    */
"description.controller.continue.new.title" = "New defect";




/*  Strings the are used in the date selections ViewController
*/

/*  the description for the alertview that confirms the upload start    */
"date.controller.alertview.confirm.description" = "Are you sure that you want to report the maintenance?";

/*  The title of the view controller    */
"date.controller.title" = "Select a Date";

/*  The title of the button that starts the uploading when possible */
"date.controller.navigation.finish.title" = "Finish";

/*  The description for the date loading view   */
"date.controller.progress.loading.description" = "Loading available dates...";

/*  Defines the amount of employees available for a daypart */
"date.controller.daypart.detail.textlabel.text" = "%@ employee(s) available";

/*  The tableview section title which holds all available titles    */
"date.controller.tableview.section.title" = "Available Dates";

/*  The description of the error message when JSON data is corrupted    */
"date.controller.result.error.corrupted" = "The loaded data is corrupt, please try again.";

/*  The error message when no results were found, no employees are available    */
```

```
"date.controller.result.error.no.availability" = "There are no available dates at the moment, please try again
    later.";




/*  Strings for default UIButton titels */
/*  title for the next button   */
"button.title.next" = "Next";



/*  the following strings describe titles and description for action sheets */
/*  The title of the button for taking a new photo  */
"actionsheet.button.image.new.title" = "Take a new Photo";

/*  the title of the button for selecting a picture from the camera roll    */
"actionsheet.button.image.cameraroll.title" = "Select from Camera Roll";

/*  the title for cancelling the actionsheet    */
"actionsheet.button.cancel.title" = "Cancel";

/*  Cancels the current reservation selected    */
"actionsheet.button.cancel.reservation.title" = "Cancel reservation";



/*  Error messages not belonging to a specific controller   */
/*  When the operation got cancelled this description is localized  */
"operation.connection.cancel.description" = "The connection was cancelled, try again later.";



/*  Reservation status types description and other strings  */
/*  The status that the reservation is being created and / or is not finished yet   */
"reservation.status.creating.title" = "Creating";

/*  The status that the reservation is being uploaded   */
"reservation.status.uploading.title" = "Uploading";

/*  The status the the reservation is uploaded but is waiting for approval  */
"reservation.status.waiting.title" = "Waiting";

/*  The status approved reservation */
"reservation.status.approved.title" = "Approved";

/*  The stats of the reservation being denied   */
"reservation.status.denied.title" = "Denied";

/*  The status of the reservation failed to upload  */
"reservation.status.failed.title" = "Failed to upload";

/*  When a defect description string could not be found show this   */
"reservation.defect.unknown" = "Unknown";

/*  When the reservation is successfully executed   */
"reservation.status.finished.title" = "Finished";

/*  When the reservation was cancelled by the user  */
"reservation.status.cancelled.title" = "Cancelled";

/*  The edit title for cancellation */
"reservation.tableview.cancel.title" = "Cancel";

/*  The edit title for deletion */
"reservation.tableview.delete.title" = "Delete";


/*  The following strings describe UIAlertView titles and descriptions
*/

/*  Success action executed title in an alertview */
"alertview.title.success" = "Success";

/*  Failed action executed title in an alertview    */
"alertview.title.error" = "Error";

/*  the confirm title for an alert view */
"alertview.title.confirm" = "Confirm Action";

/*  The title for an error with writing an image to disk    */
```

```
"alertview.title.save.failed" = "Failed to Save";

/*  the button title for the action that dismisses the alert view   */
"alertview.button.title.dismiss" = "Dismiss";

/*  The button title for the the action that lets the user retry */
"alertview.button.title.retry" = "Retry";

/*  The button that lets the user go to a page that should be edited    */
"alertview.button.title.edit" = "Edit";

/*  The button that cancels the action that can be made with the alert  */
"alertview.button.title.cancel" = "Cancel";

/*  the button that confirms an action that will be made    */
"alertview.button.title.confirm" = "Confirm";
```

```
/* Localized versions of Info.plist keys */
"CFBundleDisplayName"    =    "Opdrachtgever";

/*  this string is shown to the user when access to the photo library is requested  */
"NSPhotoLibraryUsageDescription" = "Foto's die je in de applicatie gebruikt zullen verstuurt worden om als bewijs
    van een reparatieverzoek te dienen.";
```

```
/* Localized versions of Info.plist keys */
"CFBundleDisplayName" = "BouwCloud";

/*  this string is shown to the user when access to the photo library is requested  */
"NSPhotoLibraryUsageDescription" = "Images will be uploaded to identify a defect.";
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>en</string>
    <key>CFBundleDisplayName</key>
    <string>${PRODUCT_NAME}</string>
    <key>CFBundleExecutable</key>
    <string>${EXECUTABLE_NAME}</string>
    <key>CFBundleIdentifier</key>
    <string>Solware.${PRODUCT_NAME:rfc1034identifier}</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundleLocalizations</key>
    <array>
        <string>fr</string>
        <string>de</string>
        <string>it</string>
        <string>en</string>
    </array>
    <key>CFBundleName</key>
    <string>${PRODUCT_NAME}</string>
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleShortVersionString</key>
    <string>0.8</string>
    <key>CFBundleSignature</key>
    <string>????</string>
    <key>CFBundleVersion</key>
    <string>0.8</string>
    <key>LSApplicationCategoryType</key>
    <string></string>
    <key>LSRequiresIPhoneOS</key>
    <true/>
    <key>NSHumanReadableCopyright</key>
    <string></string>
    <key>NSMainNibFile~ipad</key>
    <string>PhoneStoryboard-iPad</string>
    <key>UIMainStoryboardFile</key>
    <string>PhoneStoryboard</string>
    <key>UIRequiredDeviceCapabilities</key>
    <array>
        <string>armv7</string>
    </array>
    <key>UIStatusBarHidden</key>
    <false/>
    <key>UIStatusBarStyle</key>
    <string>UIStatusBarStyleDefault</string>
    <key>UISupportedInterfaceOrientations</key>
    <array>
        <string>UIInterfaceOrientationPortrait</string>
    </array>
    <key>UISupportedInterfaceOrientations~ipad</key>
    <array>
        <string>UIInterfaceOrientationPortrait</string>
        <string>UIInterfaceOrientationPortraitUpsideDown</string>
        <string>UIInterfaceOrientationLandscapeLeft</string>
        <string>UIInterfaceOrientationLandscapeRight</string>
    </array>
    <key>UIViewControllerBasedStatusBarAppearance</key>
    <true/>
</dict>
</plist>
```

```
//
//  Prefix header
//
//  The contents of this file are implicitly included at the beginning of every source file.
//

#import <Availability.h>

#ifndef __IPHONE_3_0
#warning "This project uses features only available in iOS SDK 3.0 and later."
#endif

#ifdef __OBJC__
    #import <UIKit/UIKit.h>
    #import <Foundation/Foundation.h>
    #import <CoreData/CoreData.h>
#endif
```

```
//
//  SBCollectionControllerDatasource.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 19-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@class SBCollectionController;

@protocol SBCollectionControllerDelegate <NSObject>

@required

// methds indicate that an item is selected or deselected
- (void) collectionController:(SBCollectionController *) controller didSelectItemWithIdentifier:(NSNumber *)
    identifier;
- (void) collectionControllerDidDeselectCollectionItem:(SBCollectionController *) controller;

// when an error occurs during the fetching of objects
//- (void) collectionController:(SBCollectionController *) controller didFailFetchWithError:(NSError *) error;

// defines wether the controller has data available or is reloading
//- (void) collectionControllerHasDataAvailable:(SBCollectionController *) controller;
//- (void) collectionControllerDidBecomeUnavailable:(SBCollectionController *) controller;

@end
```

```
//
//  SBCompatabilityInterface.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@class SBElementCollectionViewCell;

@protocol SBCompatabilityInterface <NSObject>

@optional

// required method that initializes the view controller objects for the corresponding iOS version
- (void) initializeInterfaceOfViewController:(UIViewController *) viewController;
- (void) initializeCollectionViewCell:(SBElementCollectionViewCell *) collectionCell inFrame:(CGRect) frame;

@end
```

```
//
//  SBCustomerControllerDelegate.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@class SBPhoneCustomerViewController, SBCustomerDescriptor;

@protocol SBCustomerControllerDelegate <NSObject>

@required

// when the user selected a customer send this message
- (void) customerController:(SBPhoneCustomerViewController *) controller didSelectItem:(SBCustomerDescriptor *)
    descriptor;

// when the user pushed the done button
- (void) customerControllerDidFinishPickingCustomer:(SBPhoneCustomerViewController *) controller;

@end
```

```
//
//  SBInternetCenterDelegate.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@class SBInternetCenter;

@protocol SBInternetCenterDelegate <NSObject>

// method gets called to the delegate object when the connection status changes to a new one
- (void) internetCenter:(SBInternetCenter *) center didChangeToState:(NSInteger) newState;

@end
```

```
//
//  SBKeyboardToolbarDelegate.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 19-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@class SBKeyboardToolbar;

@protocol SBKeyboardToolbarDelegate <NSObject>

// message gets called when the keyboard left button has been pressed
- (void) keyboardToolbarLeftBarButtonTapped:(SBKeyboardToolbar *) keyboardToolbar;

// message gets called when the keyboard right button has been pressed
- (void) keyboardToolbarRightBarButtonTapped:(SBKeyboardToolbar *) keyboardToolbar;

@end
```

```
//
//  SBMaintenanceInterface.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 04-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@class Reservation;

@protocol SBMaintenanceInterface <NSObject>

@required

// when the wizard will edit or create a new proceeding this method is called, passed with nil when a new proceeding
    should be created
- (void) willShowProceedingForIndex:(NSInteger) index;
- (void) viewControllerWillCreateNewProceeding:(UIViewController *) controller;

// when the wizard changes
- (void) viewController:(UIViewController *) viewController didChangeValue:(id) newValue forProceedingKey:(NSString
    *) key;

// when the wizard will show a proceeding view, it first request the existing value for this controller and sets it
- (id) viewController:(UIViewController *) viewController proceedingValueForKey:(NSString *) key;

// datasource method that must return all SBProceeding objects for the current reservation
- (Reservation *) viewControllerWillShowReservationOverview:(UIViewController *) viewController;

// when the reservation is ready to be uploaded this method is called
- (void) viewControllerDidFinishCreatingReservation:(UIViewController *) viewController;

@end
```

```
//
//  SBManagedObjectInterface.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 19-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@protocol SBManagedObjectInterface <NSObject>

// the following methods are required for a CoreData NSManagedObject in this application
@required

// returns the description used in a colection View tile
- (NSString *) collectionViewDescription;
- (NSNumber *) nextCategoryIdentifier;

// sets or retrieves the image object for the managed object
- (void) setManagedObjectImage:(UIImage *) image;
- (UIImage *) managedObjectImage;

// set or unset whether an image was loaded
- (void) setImageLoaded:(BOOL) yesNo;
- (BOOL) isImageLoaded;

// returns the number for the object ID used in the external database
- (NSNumber *) externalObjectID;

// set wether the collection view item represented for this object is selected
- (BOOL) isSelected;
- (void) setIsSelected:(BOOL) yesNo;

@end
```

```
//
//  SBOperationDelegateNew.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@class SBOperation, SBOperationStateInterface;

@protocol SBOperationDelegate <NSObject>

@required

// when an operation failed to execute with the provided error object
- (void) operation:(SBOperation *) operation didFailWithError:(NSError *) error;

@optional

// when the operation changed to a new state
- (void) operation:(SBOperation *) operation didChangeToState:(SBOperationStateInterface *) newState;

@end
```

```
//
//  SBProcessorObserverDelegate.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 27-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@class SBProcessorObserver;

@protocol SBProcessorObserverDelegate <NSObject>

// gets called when the processing perentage that is used changes
- (void) processorObserver:(SBProcessorObserver *) observer didObserveProcessorUsage:(float) usage;

@end
```

```
//
//  SBReservationManagerDelegate.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@class SBReservationManager;

@protocol SBReservationManagerDelegate <NSObject>

// method indicates that the reservation delegate is currently executing uploads or refresh
- (void) manager:(SBReservationManager *) manager didUpdateActivityWithDescription:(NSString *) description;

// method indicates that all activity has ended
- (void) managerDidEndActivity:(SBReservationManager *) manager;

@end
```

```
//
//  SBURLOperationDelegate.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

// define classes
@class SBOperation, SBAPIResponse;

@protocol SBURLOperationDelegate <NSObject>

@required

// when the operation finished downloading and create the corresponding response object
- (void) operation:(SBOperation *) operation didFinishWithResult:(SBAPIResponse *) response;

@optional

// when the operation finishes with a provided image, the options of the operation should define which object the
    image belongs to
- (void) operation:(SBOperation *) operation didFinishWithImage:(UIImage *) image;

// gets called when more bytes have been written to the socket
- (void) operation:(SBOperation *) operation didWriteBytes:(NSInteger) writtenBytes ofTotalBytes:(NSInteger)
    totalBytes;

// gets called when a new portion of the received bytes have been loaded
- (void) operation:(SBOperation *) operation didReceiveBytes:(NSInteger) receivedBytes ofExpectedBytes:(NSInteger)
    expectedBytes;

@end
```

```
//
//  SBAPIRequest.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

// define all available API request types
static NSString *SBAPIAllCustomersRequestType       =   @"allCustomers/";
static NSString *SBAPIAuthorizationRequestType      =   @"authenticationRequest/";
static NSString *SBAPIAddMaintenanceRequestType     =   @"addMaintenance/";
static NSString *SBAPIAddressLookupRequestType      =   @"lookUpAddress/";
static NSString *SBAPIAddUserRequestType            =   @"addUser/";
static NSString *SBAPIAvailableDatesRequestType     =   @"getAvailableDates/";
static NSString *SBAPICancelReservationRequestType  =   @"cancelReservation/";
static NSString *SBAPIGetImageRequestType           =   @"getImage/";
static NSString *SBAPISynchronizeDefectsRequestType =   @"synchronizeDefects/";
static NSString *SBAPICheckStateRequestType         =   @"checkState/";

// define the test, production and development server addresses
static NSString *SBAPIDevelopmentServerURL          =   @"http://stephan.bc-dev.dev.solware.local/api";
static NSString *SBAPITestServerURL                 =   @"https://bc-test.solware.nl/api";
static NSString *SBAPIProductionServerURL           =   @"";

typedef NS_ENUM(NSUInteger, SBHTTPMethod) {
    SBHTTPMethodPOST    =   1,
    SBHTTPMethodGET     =   2,
    SBHTTPMethodDELETE  =   3,
    SBHTTPMethodHEAD    =   4,
    SBHTTPMethodPUT     =   5,
    SBHTTPMethodTRACE   =   6,
};

@interface SBAPIRequest : NSObject
{
    // whether the HTTP method has been custom set
    BOOL _isHTTPMethodDefined;

    // the body used for the request
    NSMutableData *_requestBody;

    // the HTTP method used for the request
    SBHTTPMethod _requestMethod;
}

// the boundary that will be used to seperate POST data objects and the URL that will be requested
@property (nonatomic, strong, readonly) NSString *boundary;

// creates a new API request object wth the provided URL type
- (SBAPIRequest *) initWithRequestCall:(NSString *) callName timeout:(NSTimeInterval) timeoutInterval;

// adds the provided object data and key name to the POST data of the request
- (void) setPOSTValue:(id) object forKey:(NSString *) POSTKey;
- (void) addPOSTFileData:(NSData *) fileData withFileName:(NSString *) fileName MIMEType:(NSString *) MIME;

// retrieves the final request that will be used
- (NSURLRequest *) finalURLRequest;

// retrieves the URL used for the request or other request values
- (NSURL *) URL;
- (NSData *) requestBody;

// sets the HTTP method of the request, the method will be set by the finalURLRequest method when no method has been
    defined
- (void) setHTTPMethod:(SBHTTPMethod) newMethod;
- (SBHTTPMethod) HTTPMethod;

// sets a new HTTP header for this request
- (void) setHTTPHeaderValue:(NSString *) value forKey:(NSString *) key;

// retrieve all HTTP headers used by the request or a single value for the given key
- (NSString *) HTTPHeaderValueForKey:(NSString *) key;
- (NSDictionary *) allHTTPHeaders;

@end
```

```
//
//  SBAPIRequest.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBAPIRequest.h"
#import "SBVerificationManager.h"

@interface SBAPIRequest()

// creates a new boundary object used for POST data
- (NSString *) boundaryWithLength:(NSInteger) length;

// the used URL request for this API call
@property (nonatomic, strong) NSMutableURLRequest *URLRequest;

@end

// define the version of the API that will be used
#define kSBAPIVersionIdentifier   "v1"

@implementation SBAPIRequest

/*  Initializes the API request object with the provided URL call name and timeout  */
- (SBAPIRequest *) initWithRequestCall:(NSString *) callName timeout:(NSTimeInterval) timeoutInterval
{
    self = [super init];

    if (self) {

        // set the boundary that will be used
        _boundary = [self boundaryWithLength: 16];
        _requestBody = [NSMutableData dataWithLength: 0];
        _requestMethod = SBHTTPMethodGET;

        // create the URL object from the provided information
        NSURL *requestURL = [NSURL URLWithString: [NSString stringWithFormat: @"%@/%s/%@", SBAPITestServerURL,
            kSBAPIVersionIdentifier, callName]];

        // generate the URL request, no cache will be used by default. This will be handled by the operation classes
            when neccesary such as image caching
        self.URLRequest = [[NSMutableURLRequest alloc] initWithURL: requestURL cachePolicy:
            NSURLRequestReloadIgnoringLocalCacheData timeoutInterval: timeoutInterval];
    }

    return self;
}

/*  Overrides the URL request getter method to do some last changes before the request will be sent */
- (NSURLRequest *) finalURLRequest
{
    // when the data container is set make sure that the request HTTP type is "POST"
    if ([_requestBody length] > 0) {

        // set the HTTP method to POST when no HTTP method has been set
        if (!_isHTTPMethodDefined) {
            _requestMethod = SBHTTPMethodPOST;
        }

        // seth the content header to multipart form-data
        [self.URLRequest setValue: [NSString stringWithFormat: @"multipart/form-data, boundary=%@", self.boundary]
            forHTTPHeaderField: @"Content-Type"];

        // set the HTTP body data after appending the final delimiter
        [_requestBody appendData: [[NSString stringWithFormat: @"--%@--\r\n", self.boundary] dataUsingEncoding:
            NSUTF8StringEncoding]];
        [self.URLRequest setHTTPBody: _requestBody];

        // finally set the content length header
        [self.URLRequest setValue: [NSString stringWithFormat: @"%d", [_requestBody length]] forHTTPHeaderField:
            @"Content-Length"];
    }

    // apply the HTTP method
    if (!_isHTTPMethodDefined) {
        [self setHTTPMethod: _requestMethod];
    }

    // last set the verification credentials as header
    [self.URLRequest setValue: [NSString stringWithFormat: @"Basic %@", [[SBVerificationManager defaultVerification]
        generateBase64AuthorizationString]] forHTTPHeaderField: @"Authorization"];
    NSLog(@"Authorization: %@", [self.URLRequest valueForHTTPHeaderField: @"Authorization"]);

    // return the final request that can be sent
```

```
    return self.URLRequest;
}

/*  Return the URL that is used for this request    */
- (NSURL *) URL
{
    // return the request's URL
    return self.URLRequest.URL;
}

/*  Create a new boundary object for the provided length    */
- (NSString *) boundaryWithLength:(NSInteger)length
{
    // the string with all possible characters in the boundary
    NSString *boundaryCharacters = @"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    NSMutableString *randomBoundary = [NSMutableString stringWithCapacity: length];

    // add a random character for each loop
    for (int i = 0; i < length; i++) {
        [randomBoundary appendFormat: @"%C", [boundaryCharacters characterAtIndex: (arc4random() %
            [boundaryCharacters length])]];
    }

    // return the generated boundary
    return randomBoundary;
}

/*  Adds a new data value and key pair to the POST data of the URL request  */
- (void) setPOSTValue:(id) object forKey:(NSString *) POSTKey
{
    // create a mutable data container to add to the current POST data
    NSMutableData *dataContainer = [NSMutableData dataWithLength: 0];

    // add the boundary tag to the container and the form data name
    [dataContainer appendData: [[NSString stringWithFormat: @"--%@\r\n", self.boundary] dataUsingEncoding:
        NSUTF8StringEncoding]];
    [dataContainer appendData: [[NSString stringWithFormat: @"Content-disposition: form-data; name=\"%@\"\r\n\r\n",
        POSTKey] dataUsingEncoding: NSUTF8StringEncoding]];

    // add the data value to the container and commit the created data
    [dataContainer appendData: [[NSString stringWithFormat: @"%@\r\n", object] dataUsingEncoding:
        NSUTF8StringEncoding]];
    [_requestBody appendData: dataContainer];
}

/*  Adds a new file data container with the provided file name to the request that will be sent */
- (void) addPOSTFileData:(NSData *) fileData withFileName:(NSString *) fileName MIMEType:(NSString *) MIME
{
    // create the mutable data container for the file data
    NSMutableData *dataContainer = [NSMutableData dataWithLength: 0];

    // add the boundary data, then set the content type for file, add the filename and MIME type
    [dataContainer appendData: [[NSString stringWithFormat: @"--%@\r\n", self.boundary] dataUsingEncoding:
        NSUTF8StringEncoding]];
    [dataContainer appendData: [[NSString stringWithFormat: @"Content-Disposition: form-data; name=\"userfile\";
        filename=\"%@\"\r\n", fileName] dataUsingEncoding: NSUTF8StringEncoding]];
    [dataContainer appendData: [[NSString stringWithFormat: @"Content-Type: %@\r\n\r\n", MIME] dataUsingEncoding:
        NSUTF8StringEncoding]];

    // add the copied bytes the the requestbody to represent the jpg image to be uploaded
    [dataContainer appendData: fileData];
    [dataContainer appendData: [[NSString stringWithFormat: @"\r\n"] dataUsingEncoding: NSUTF8StringEncoding]];

    // append the new data to the request
    [_requestBody appendData: dataContainer];
}

/*  Returns the request body that will be used for the body of the request, this body is possible not to be complete
    when the final URL request has not been called */
- (NSData *) requestBody
{
    return _requestBody;
}

/*  Sets the new HTTP method of the url request */
- (void) setHTTPMethod:(SBHTTPMethod)newMethod
{
    // set the corresponding HTTP method for the request
    switch (newMethod) {
        case SBHTTPMethodDELETE:
            [self.URLRequest setHTTPMethod: @"DELETE"];
            break;

        case SBHTTPMethodGET:
            [self.URLRequest setHTTPMethod: @"GET"];
            break;
```

```
        case SBHTTPMethodHEAD:
            [self.URLRequest setHTTPMethod: @"HEAD"];
            break;

        case SBHTTPMethodPOST:
            [self.URLRequest setHTTPMethod: @"POST"];
            break;

        case SBHTTPMethodPUT:
            [self.URLRequest setHTTPMethod: @"PUT"];
            break;

        case SBHTTPMethodTRACE:
            [self.URLRequest setHTTPMethod: @"TRACE"];
            break;
    }

    // set that the HTTP method has been custom set
    _isHTTPMethodDefined = YES;
    _requestMethod = newMethod;
}

/*  Retrieve the current HTTP method used for this request  */
- (SBHTTPMethod) HTTPMethod
{
    return _requestMethod;
}

/*  Sets the given HTTP header value for the given key  */
- (void) setHTTPHeaderValue:(NSString *)value forKey:(NSString *)key
{
    // apply header change to the request
    [self.URLRequest setValue: value forHTTPHeaderField: key];
}

/*  Returns all header fields of the request that will be sent  */
- (NSDictionary *) allHTTPHeaders
{
    return [self.URLRequest allHTTPHeaderFields];
}

/*  Returns a single string value for the given HTTP header key */
- (NSString *) HTTPHeaderValueForKey:(NSString *)key
{
    return [self.URLRequest valueForHTTPHeaderField: key];
}

@end
```

```
//
//  SBAPIResult.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 06-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

// the different types of results for the API result
typedef enum {
    SBAPIResultSuccessType      =   1,
    SBAPIResultFailType         =   2,
    SBAPIResultErrorType        =   3,
    SBAPIResultUnknownType      =   4,
} SBAPIResultType;

// string key identifiers for necessary objects
static NSString * SBAPIResultStatusKey      =   @"status";
static NSString * SBAPIResultMessageKey     =   @"message";
static NSString * SBAPIResultCodeKey        =   @"code";
static NSString * SBAPIResultDataKey        =   @"data";

// the domain for the JSend error objects
static NSString * SBJSendErrorDomain        =   @"JSendErrorDomain";

// define the code used for identifiying invalid JSend object
#define kSBAPIResultInvalidData     1

// defines all possible return codes of the status
typedef NS_ENUM(NSUInteger, SBHTTPResponseCode) {
    SBHTTPResponseCodeOK                    =   200,
    SBHTTPResponseCodePermanently           =   301,
    SBHTTPResponseCodeFound                 =   302,
    SBHTTPResponseCodeNotModified           =   304,
    SBHTTPResponseCodeTemporaryRedirect     =   307,
    SBHTTPResponseCodeBadRequest            =   400,
    SBHTTPResponseCodeUnauthorized          =   401,
    SBHTTPResponseCodeForbidden             =   403,
    SBHTTPResponseCodeNotFound              =   404,
    SBHTTPResponseCodeConflict              =   409,
    SBHTTPResponseCodeServerError           =   500,
};

@interface SBAPIResponse : NSObject
{
    // the URL response object
    NSURLResponse *_URLResponse;
}

// the result type for this object received from the API server
@property (nonatomic) SBAPIResultType objectResultType;

// the dictionary with result data, can be an array or dictionary etc
@property (nonatomic, strong, readonly) id resultDictionary;
@property (nonatomic, strong, readonly) UIImage *imageResource;

// the message provided in the result JSend object
@property (nonatomic, strong, readonly) NSString *resultMessage;

// the code provided when an error occured during processing on the server
@property (nonatomic, strong, readonly) NSNumber *resultCode;

// method initializes the API result object with the downloaded data
- (SBAPIResponse *) initWithURLResponse:(NSURLResponse *) response;

// the parsing method which validates the data set as JSON or image
- (BOOL) parseData:(NSData *) downloadedData withError:(NSError **) error;

// returns whether the received data represents a file
- (BOOL) isImage;

// retrieves the status code of the response
- (SBHTTPResponseCode) responseStatusCode;

@end
```

```
//
//  SBAPIResult.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 06-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBAPIResponse.h"

// expected mime types
static NSString *SBJPEGImageAPIResponseMIMEType       =   @"image/jpeg";
static NSString *SBPNGImageAPIResponseMIMEType        =   @"image/png";
static NSString *SBJSONAPIResponseMIMEType            =   @"application/json";

@interface SBAPIResponse()

// internaly parse the received data as JSON object
- (BOOL) parseJSONData:(NSData *) data withError:(NSError **) error;

// internally parse the received data as image resource
- (BOOL) parseImageData:(NSData *) data;

@end

@implementation SBAPIResponse

/*  Create the new api result object with the downloaded data   */
- (SBAPIResponse *) initWithURLResponse:(NSURLResponse *) response
{
    // create super object
    self = [super init];

    if (self) {

        // set the response object
        _URLResponse = response;
        self.objectResultType = SBAPIResultUnknownType;
    }

    return self;
}

/*  Returns whether the fetched data represents a file or other resource    */
- (BOOL) isImage
{
    // when the MIME type of the URL response is either a jpeg or png the resource downloaded is an image
    if ([[_URLResponse MIMEType] isEqualToString: SBJPEGImageAPIResponseMIMEType] || [[_URLResponse MIMEType]
        isEqualToString: SBPNGImageAPIResponseMIMEType]) {

        // the downloaded content is an image so return YES
        return YES;
    } else {
        return NO;
    }
}

/*  Parses the downloaded data and checks whether the resource is an image or JSON object   */
- (BOOL) parseData:(NSData *) downloadedData withError:(NSError *__autoreleasing *) error
{
    // when an image should be received parse as image
    if ([self isImage]) {

        // parse as image and return the result
        if (![self parseImageData: downloadedData]) {

            // create the error descirption
            NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObjects:
                NSLocalizedString(@"error.response.image.corrupt.description", nil), nil] forKeys: [NSArray
                arrayWithObjects: NSLocalizedDescriptionKey, nil]];

            // an error occured during the parsing of the image
            *error = [NSError errorWithDomain: NSCocoaErrorDomain code: -1 userInfo: userInfo];
            return NO;
        } else {

            // successfully parsed the image, also the method has set the resource image in the response
            return YES;
        }
    } else {

        // parse the received data as JSON object and return its result
        return [self parseJSONData: downloadedData withError: error];
    }
}

/*  Parses the received data as an image, sets the error message when not successfully. Also the returned image when
```

```
    set is always a PNG for performance, all images will be converted when needed  */
- (BOOL) parseImageData:(NSData *) data
{
    // create an image with the provided data for the screen scale
    UIImage *newImage = [UIImage imageWithData: data scale: [[UIScreen mainScreen] scale]];
    BOOL returnValue = NO;

    // when the image could be created from the data continue
    if (newImage != nil) {

        // when the image is not yet a PNG representation the image should be parsed
        if (![[_URLResponse MIMEType] isEqualToString: SBPNGImageAPIResponseMIMEType]) {

            // get the data as a PNG image
            NSData *PNGData = UIImagePNGRepresentation(newImage);
            _imageResource = [UIImage imageWithData: PNGData scale: [[UIScreen mainScreen] scale]];

            // check whether the image could be created
            if (self.imageResource != nil) {
                returnValue = YES;
            }
        } else {

            // return true and set the created image
            _imageResource = newImage;
            returnValue = YES;
        }
    }

    return returnValue;
}

/*  Start the parsing of the data provided in the initializer, when failed NO is returned and the error object will
    be set  */
- (BOOL) parseJSONData:(NSData *) data withError:(NSError *__autoreleasing *) error
{
    // start creating a dictionary representation of the provided data
    NSError *parseError = nil;
    BOOL returnValue = NO;
    NSDictionary *tempResult = [NSJSONSerialization JSONObjectWithData: data options: NSJSONReadingAllowFragments
        error: &parseError];

    // when no error occured continue by validating the code, message etc.
    if (parseError == nil) {

        // check for the status to be provided
        id responseStatus = [tempResult objectForKey: SBAPIResultStatusKey];

        // when the status is set set the appropriate enumeration value
        if (responseStatus != nil && [responseStatus isKindOfClass: [NSString class]]) {

            // validate the enumeration type between success, fail and error. Each different value holds different
                types of dictionary values.
            if ([responseStatus isEqualToString: @"success"]) {
                self.objectResultType = SBAPIResultSuccessType;
            } else if ([responseStatus isEqualToString: @"fail"]) {
                self.objectResultType = SBAPIResultFailType;
            } else if ([responseStatus isEqualToString: @"error"]) {
                self.objectResultType = SBAPIResultErrorType;

                // get the message result object
                id tempResultMessage = [tempResult objectForKey: SBAPIResultMessageKey];
                id tempResultCode = [tempResult objectForKey: SBAPIResultCodeKey];

                // set the message when found and is is a string object
                if (tempResultMessage != nil && [tempResultMessage isKindOfClass: [NSString class]]) {
                    _resultMessage = (NSString *)tempResultMessage;
                }

                // set the error code when found
                if (tempResultCode != nil && [tempResultCode isKindOfClass: [NSNumber class]]) {
                    _resultCode = (NSNumber *)tempResultCode;
                }
            } else {

                // create the error descirption
                NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObjects:
                    NSLocalizedString(@"error.response.invalid.response.description", nil), _URLResponse.URL, nil]
                    forKeys: [NSArray arrayWithObjects: NSLocalizedDescriptionKey, NSURLErrorKey, nil]];

                // no valid result type is found, create error object and return false
                *error = [NSError errorWithDomain: SBJSendErrorDomain code: -1 userInfo: userInfo];
                self.objectResultType = SBAPIResultUnknownType;
            }

            // when the result type is defined the data can be parsed
            if (self.objectResultType != SBAPIResultUnknownType) {
```

```
                // when the data is set set the result dictionary
                _resultDictionary = [tempResult objectForKey: SBAPIResultDataKey];
                returnValue = YES;
            }
        } else {

            // create the error descirption
            NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObjects:
                NSLocalizedString(@"error.response.invalid.response.description", nil), _URLResponse.URL, nil]
                forKeys: [NSArray arrayWithObjects: NSLocalizedDescriptionKey, NSURLErrorKey, nil]];

            // create the error object which defines the missing success key
            *error = [NSError errorWithDomain: SBJSendErrorDomain code: -1 userInfo: userInfo];
        }
    } else {

        // set the error object that was created during the parsing
        *error = parseError;
    }

    return returnValue;
}

/*  Returns the URL status code for the response    */
- (SBHTTPResponseCode) responseStatusCode
{
    // return the status code of the HTTP url response
    return [(NSHTTPURLResponse *)_URLResponse statusCode];
}

@end
```

```
//
//  SBAvailableDateDescriptor.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 05-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <EventKit/EventKit.h>

static NSString const * SBAvailableDateDescriptorObjectsKey         =   @"data";
static NSString const * SBAvailableDateDescriptorDayPartKey         =   @"day_part";
static NSString const * SBAvailableDateDescriptorResultKey          =   @"success";
static NSString const * SBAvailableDateDescriptorDateKey            =   @"date";
static NSString const * SBAvailableDateDescriptorDescriptionKey     =   @"description";
static NSString const * SBAvailableDateDescriptorDayPartIdKey       =   @"id";
static NSString const * SBAvailableDateDescriptorStartTimeKey       =   @"start";
static NSString const * SBAvailableDateDescriptorEndTimeKey         =   @"end";
static NSString const * SBAvailableDateDescriptorCountKey           =   @"available";

@class SBDayPartDescriptor;

@interface SBAvailableDateDescriptor : NSObject

@property (nonatomic, strong) NSDate *availableDate;
@property (nonatomic, strong) NSMutableArray *availableTime;

- (SBAvailableDateDescriptor *) initWithDictionary:(NSDictionary *) dictionary;

// returns the amount of time available objects there are
- (NSInteger) totalAvailableTimes;
- (void) resetAllTimeIndication;

- (SBDayPartDescriptor *) selectedDayPart;
- (void) processEvent:(EKEvent *) event withCalendar:(NSCalendar *) calendar;

@end
```

```
//
//  SBAvailableDateDescriptor.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 05-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBAvailableDateDescriptor.h"
#import "SBDayPartDescriptor.h"
#import <EventKit/EventKit.h>

@implementation SBAvailableDateDescriptor

/*  Initializes the date discrpitor with a JSON dictionary array object, contents are a date and one or more
    dayparts for which the user can subscribe  */
- (SBAvailableDateDescriptor *) initWithDictionary:(NSDictionary *)dictionary
{
    self = [super init];

    if (self)
    {
        _availableTime = [[NSMutableArray alloc] initWithCapacity: 4];

        if ([dictionary objectForKey: SBAvailableDateDescriptorDateKey] != nil)
        {
            // set the date of this availability descriptor
            NSTimeInterval interval = [[dictionary objectForKey: SBAvailableDateDescriptorDateKey] doubleValue];
            _availableDate = [NSDate dateWithTimeIntervalSince1970: interval];

            // get the array when available
            NSArray *dayParts = [dictionary objectForKey: SBAvailableDateDescriptorDayPartKey];
            SBDayPartDescriptor *dayDescriptor = nil;
            NSNumberFormatter *numberFormatter = [[NSNumberFormatter alloc] init];

            // create the date formatter for the day part formatting
            NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
            NSCalendar *gregorianCalendar = [[NSCalendar alloc] initWithCalendarIdentifier: NSGregorianCalendar];
            [dateFormatter setDateFormat: @"H:m:s"];

            for (int i = 0; i < [dayParts count]; i++)
            {
                // create a time descriptor object to add to the array
                dayDescriptor = [[SBDayPartDescriptor alloc] initWithID: [numberFormatter numberFromString:
                    [[dayParts objectAtIndex: i] objectForKey: SBAvailableDateDescriptorDayPartIdKey]]
                    andDescription: [[dayParts objectAtIndex: i] objectForKey:
                    SBAvailableDateDescriptorDescriptionKey]];
                [_availableTime addObject: dayDescriptor];

                // set the availability counter
                [dayDescriptor setAvailableCount: [[dayParts objectAtIndex: i] objectForKey:
                    SBAvailableDateDescriptorCountKey]];

                // set the start and end date
                NSDate *startDate = [dateFormatter dateFromString: [[dayParts objectAtIndex: i] objectForKey:
                    SBAvailableDateDescriptorStartTimeKey]];
                NSDate *endDate = [dateFormatter dateFromString: [[dayParts objectAtIndex: i] objectForKey:
                    SBAvailableDateDescriptorEndTimeKey]];

                // add the dates from the JSON data to the day part object
                [dayDescriptor setStartTime: [gregorianCalendar components: NSHourCalendarUnit fromDate: startDate]]
                    ;
                [dayDescriptor setEndTime: [gregorianCalendar components: NSHourCalendarUnit fromDate: endDate]];
            }
        }
    }

    return self;
}

/*  Returns the total amount of time spans available for this date  */
- (NSInteger) totalAvailableTimes
{
    return [_availableTime count];
}

/*  Reset all dayparts to not selected  */
- (void) resetAllTimeIndication
{
    // for each available time set selected to NO
    for (int i = 0; i < [_availableTime count]; i++)
        [[_availableTime objectAtIndex: i] setIsSelected: NO];
}

/*  Returns the day part id of the currently selected daypart   */
- (SBDayPartDescriptor *) selectedDayPart
{
```

```
    // loop through the objects to find the selected time ID
    for (int i = 0; i < [_availableTime count]; i++)
    {
        // when the time is selected return its Number
        if ([[_availableTime objectAtIndex: i] isSelected])
            return [_availableTime objectAtIndex: i];
    }

    return nil;
}

/*  Processes the event, checks wether the date of the event is the same, when equel process it deeper to the time
    */
- (void) processEvent:(EKEvent *) event withCalendar:(NSCalendar *) calendar
{
    // get the events start and end date components, also retreive the date components of this objects date
    NSDateComponents *eventStartDate = [calendar components: (NSYearCalendarUnit | NSMonthCalendarUnit |
        NSDayCalendarUnit) fromDate: [event startDate]];
    NSDateComponents *eventEndDate = [calendar components: (NSYearCalendarUnit | NSMonthCalendarUnit |
        NSDayCalendarUnit) fromDate: [event endDate]];
    NSDateComponents *currentDate = [calendar components: (NSYearCalendarUnit | NSMonthCalendarUnit |
        NSDayCalendarUnit) fromDate: _availableDate];

    // when the current date components is equal to the start or end date continue
    if ([currentDate isEqual: eventStartDate] || [currentDate isEqual: eventEndDate])
    {
        // get the hours from the event dates
        NSInteger eventStartHour = [[calendar components: NSHourCalendarUnit fromDate: [event startDate]] hour];
        NSInteger eventEndHour = [[calendar components: NSHourCalendarUnit fromDate: [event endDate]] hour];

        // get the daypart enumerator
        NSEnumerator *dayPartEnumerator = [_availableTime objectEnumerator];
        SBDayPartDescriptor *currentDescriptor = nil;

        // loop through all day parts and let the descriptor determine if the daypart matches the event
        while (currentDescriptor = [dayPartEnumerator nextObject])
            [currentDescriptor processEvent: event withStartHour: eventStartHour endHour: eventEndHour];
    }
}

@end
```

```
//
//  SBCollectionCellViewLayout.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 12-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

// the identifier for the supplementary title view used in the collection cell
static NSString *SBCollectionViewLayoutItemKindTitle            =   @"ItemKindView";
static NSString *SBCollectionViewLayoutSupplementaryKindTitle   =   @"SupplemantoryTitleView";

@interface SBCollectionCellViewLayout : UICollectionViewLayout

// the default height for the title string
@property (nonatomic) CGFloat titleHeight;
@property (nonatomic) NSInteger maxRowCount;
@property (nonatomic) UIEdgeInsets cellEdgeInsets;
@property (nonatomic) CGSize itemSize;

// the weak reference to the use fetched results controller, this way object sizes can be calculated
@property (nonatomic, weak) NSFetchedResultsController *collectionFetchedResultsController;

// the dictionary with information about all layout attributes
@property (nonatomic, strong) NSMutableDictionary *layoutAttributesInfo;

@end
```

```
//
//  SBCollectionCellViewLayout.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 12-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBCollectionCellViewLayout.h"

@interface SBCollectionCellViewLayout()

// the initialize method called by all initialization methods
- (void) initialize;

// calculate the frame for the object at the given indexpath
- (CGRect) calculateFrameForObjectAtIndexPath:(NSIndexPath *) indexPath;

// creates all needed attributes layout for the items
- (void) prepareAttributesLayout;

@end


@implementation SBCollectionCellViewLayout

/*  The normal initializer method   */
- (SBCollectionCellViewLayout *) init
{
    self = [super init];

    // setup the object
    if (self) {
        [self initialize];
    }

    return self;
}

/*  When initialized with a Decoder also initialize the object  */
- (id) initWithCoder:(NSCoder *)aDecoder
{
    self = [super initWithCoder: aDecoder];

    // setup the object
    if (self) {
        [self initialize];
    }

    return self;
}

/*  The default initialization method, sets the height etc, */
- (void) initialize
{
    // set default values
    self.titleHeight = 50;
    self.maxRowCount = 3;
    self.cellEdgeInsets = UIEdgeInsetsMake(5, 10, 5, 10);
    self.itemSize = CGSizeMake(90, 90);

    // create the dictionary info container
    self.layoutAttributesInfo = [NSMutableDictionary dictionary];
}

/*  Returns the content view size, this is the size of the full view including supplementary    */
- (CGSize) collectionViewContentSize
{
    // get the amount of rows in the collection view
    NSInteger totalRows = [self.collectionView numberOfItemsInSection: 0] / self.maxRowCount;

    // when a row is not completly filled with the max amount of objects add a extra row, the row will not be shown
        otherwise
    if ([self.collectionView numberOfItemsInSection: 0] % self.maxRowCount) {
        totalRows++;
    }

    // calculate the height of all objects combined that need to be showed
    CGFloat totalHeight = (self.cellEdgeInsets.top + self.cellEdgeInsets.bottom + self.itemSize.height + self.
        titleHeight) * totalRows;

    // return the appropriate frame
    return CGSizeMake(self.collectionView.bounds.size.width, totalHeight);
}

/*  Gets called when elements should be layed out in the specified rectangle, returns an array of attributes    */
- (NSArray *) layoutAttributesForElementsInRect:(CGRect)rect
```

```
{
    // create all attributes objects when it is empty
    if ([self.layoutAttributesInfo count] == 0) {
        [self prepareAttributesLayout];
    }

    // create the mutable array with capacity for all attributes
    NSMutableArray *attributeArray = [NSMutableArray arrayWithCapacity: [self.layoutAttributesInfo count]];

    // get all layout attributes for all types defined in the dictionary
    [self.layoutAttributesInfo enumerateKeysAndObjectsUsingBlock: ^(NSString *elementIdentifierKey, NSDictionary *
        elementSupplemantoryViewValues, BOOL *stop) {

        // enumerate trouhg all objects and keys of the dictionary
        [elementSupplemantoryViewValues enumerateKeysAndObjectsUsingBlock: ^(NSIndexPath *indexPathKey,
            UICollectionViewLayoutAttributes *layoutAttributesValue, BOOL *stop) {

            // when the current object is available in the given rect add it to the array
            if (CGRectIntersectsRect(rect, layoutAttributesValue.frame)) {
                [attributeArray addObject: layoutAttributesValue];
            }
        }];
    }];

    // return the created array
    return attributeArray;
}

/*  Create all item attribute objects for the custom layout */
- (void) prepareAttributesLayout
{
    // create the used array container for the attributes
    NSMutableDictionary *layoutAttributes = [NSMutableDictionary dictionary];
    NSMutableDictionary *titleLayoutAttributes = [NSMutableDictionary dictionary];

    // get the amount of objects
    NSInteger objectCount = [self.collectionView numberOfItemsInSection: 0];
    NSIndexPath *currentIndexpath = nil;

    for (int i = 0; i < objectCount; i++) {

        // set the new current indexpath
        currentIndexpath = [NSIndexPath indexPathForItem: i inSection: 0];

        // create the attributes for the current indexpath
        UICollectionViewLayoutAttributes *itemAttributes = [UICollectionViewLayoutAttributes
            layoutAttributesForCellWithIndexPath: currentIndexpath];

        // set the layout frame and add the layout to the dictionary container
        CGRect currentIndexPathFrame = [self calculateFrameForObjectAtIndexPath: currentIndexpath];
        [itemAttributes setFrame: currentIndexPathFrame];
        [layoutAttributes setObject: itemAttributes forKey: currentIndexpath];

        // create the supplementary view attribute for the item title label
        UICollectionViewLayoutAttributes *supplementaryTitleViewLayout = [UICollectionViewLayoutAttributes
            layoutAttributesForSupplementaryViewOfKind: SBCollectionViewLayoutSupplementaryKindTitle withIndexPath:
            currentIndexpath];

        // calculate the frame for the supplementary view
        [supplementaryTitleViewLayout setFrame: CGRectMake(currentIndexPathFrame.origin.x, currentIndexPathFrame.
            origin.y + currentIndexPathFrame.size.height, currentIndexPathFrame.size.width, self.titleHeight)];
        [titleLayoutAttributes setObject: supplementaryTitleViewLayout forKey: currentIndexpath];
    }

    // set the layout attributes for the default kind
    [self.layoutAttributesInfo setObject: layoutAttributes forKey: SBCollectionViewLayoutItemKindTitle];
    [self.layoutAttributesInfo setObject: titleLayoutAttributes forKey: SBCollectionViewLayoutSupplementaryKindTitle
        ];
}

/*  Method calculates the frame for the object at the given indexpath   */
- (CGRect) calculateFrameForObjectAtIndexPath:(NSIndexPath *)indexPath
{
    // calculate the current column and row the object will be placed in
    NSInteger row = indexPath.item / self.maxRowCount;
    NSInteger column = indexPath.item % self.maxRowCount;

    // calculate the x position for the current indexpath object
    CGFloat spacingX = (self.collectionView.bounds.size.width - self.cellEdgeInsets.left - self.cellEdgeInsets.right
        - (self.maxRowCount * self.itemSize.width)) / (self.maxRowCount - 1);

    // now calculate the actual origin coordinates with cell edge insets values calculated with it
    CGFloat originX = floorf(self.cellEdgeInsets.left + (self.itemSize.width + spacingX) * column);
    CGFloat originY = floor(self.cellEdgeInsets.top + (self.itemSize.height + self.titleHeight + 5) * row);

    // return the corresponding frame for this object
    return CGRectMake(originX, originY, self.itemSize.width, self.itemSize.height);
```

```
}

/*  Gets called when the attributes should be layed out for the specified object at indexpath   */
- (UICollectionViewLayoutAttributes *) layoutAttributesForItemAtIndexPath:(NSIndexPath *)indexPath
{
    // return the created layout object from the dictionary
    return [[self.layoutAttributesInfo objectForKey: SBCollectionViewLayoutItemKindTitle] objectForKey: indexPath];
}

/*  Returns the layout attributes for the supplementatyr view (Title view) for the object at the given index    */
- (UICollectionViewLayoutAttributes *) layoutAttributesForSupplementaryViewOfKind:(NSString *)kind atIndexPath:
    (NSIndexPath *)indexPath
{
    // return the precalculated supplementary view layout info
    return [[self.layoutAttributesInfo objectForKey: SBCollectionViewLayoutSupplementaryKindTitle] objectForKey:
        indexPath];
}

/*  Returns whether the layout should be invalideted and redrawn on first sight when new bounds are available   */
- (BOOL) shouldInvalidateLayoutForBoundsChange:(CGRect)newBounds
{
    return NO;
}

@end
```

```
//
//  SBColorManager.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface SBColorManager : NSObject

// different colors for styles in the entire application, this one is for the background in view controllers
@property (nonatomic, strong) UIColor *viewControllerBackgroundColor;

// this color is used for the request adding navigation controllers
@property (nonatomic, strong) UIColor *requestNavigationColor;

// the color used for the my info navigation controllers
@property (nonatomic, strong) UIColor *myInformationNavigationColor;

// different colors used in the overview table for status indication
@property (nonatomic, strong) UIColor *greenColor;
@property (nonatomic, strong) UIColor *orangeColor;
@property (nonatomic, strong) UIColor *yellowColor;
@property (nonatomic, strong) UIColor *redColor;

// retrieve the default color manager
+ (SBColorManager *) defaultColors;

@end
```

```
//
//  SBColorManager.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBColorManager.h"

@implementation SBColorManager

// the singleton instance used by the entire application
static SBColorManager *singleton = nil;

/* Retrieves and initializes the color manager object   */
+ (SBColorManager *) defaultColors
{
    // when the color manager is not set create a new one
    if (singleton == nil) {

        // initialize the object
        singleton = [[SBColorManager alloc] init];

        // set the default colors
        [singleton setViewControllerBackgroundColor: [UIColor colorWithRed: 0.8828125 green: 0.921875 blue:
            0.91796875 alpha: 1.0]];
        [singleton setRequestNavigationColor: [UIColor colorWithRed: 0.54296875 green: 0.8359375 blue: 0.73828125
            alpha: 1.0]];
        [singleton setMyInformationNavigationColor: [UIColor colorWithRed: 0.25 green: 0.3203125 blue: 0.41015625
            alpha: 1]];
        [singleton setRedColor: [UIColor colorWithRed: 0.88671875 green: 0.375 blue: 0.296875 alpha: 1.0]];
        [singleton setOrangeColor: [UIColor colorWithRed: 0.9296875 green: 0.46484375 blue: 0.08984375 alpha: 1.0]];
        [singleton setGreenColor: [UIColor colorWithRed: 0.359375 green: 0.71484375 blue: 0.0 alpha: 1.0]];
        [singleton setYellowColor: [UIColor colorWithRed: 0.96484375 green: 0.82421875 blue: 0.08984375 alpha: 1.0]]
            ;
    }

    // return the shared singleton
    return singleton;
}

@end
```

```objc
//
//  SBcustomerDescriptor.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface SBCustomerDescriptor : NSObject

// declare a id and name for the customer
@property (nonatomic, strong) NSNumber *customerId;
@property (nonatomic, strong) NSString *customerName;

// initializes the object with the id and name
- (SBCustomerDescriptor *) initWithID:(NSNumber *) identifier andName:(NSString *) name;

@end
```

```
//
//  SBcustomerDescriptor.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBCustomerDescriptor.h"

@implementation SBCustomerDescriptor

/*  Initializes a new customer object with the given id and name    */
- (SBCustomerDescriptor *) initWithID:(NSNumber *) identifier andName:(NSString *) name
{
    self = [super init];

    if (self)
    {
        self.customerId = identifier;
        self.customerName = name;
    }

    return self;
}

@end
```

```objc
//
//  SBDayPartDescriptor.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <EventKit/EventKit.h>

@interface SBDayPartDescriptor : NSObject <NSCoding>

@property (nonatomic, strong) NSString *timeDescription;
@property (nonatomic, strong) NSNumber *timeId;
@property (nonatomic, strong) NSNumber *availableCount;
@property (nonatomic, strong) NSDateComponents *startTime;
@property (nonatomic, strong) NSDateComponents *endTime;
@property (nonatomic) BOOL isSelected;
@property (nonatomic, strong) NSString *otherEventsDescription;
@property (nonatomic) NSUInteger totalOtherEvents;

// initialize with a ID and description
- (SBDayPartDescriptor *) initWithID:(NSNumber *) timeId andDescription:(NSString *) description;
- (BOOL) toggleSelected;

// processes and check wether the event is overlapping with the daypart
- (void) processEvent:(EKEvent *)event withStartHour:(NSInteger) start endHour:(NSInteger) end;

@end
```

```
//
//  SBDayPartDescriptor.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBDayPartDescriptor.h"

@implementation SBDayPartDescriptor

/*  Initialize the time descriptor with a time and object ID    */
- (SBDayPartDescriptor *) initWithID:(NSNumber *)timeId andDescription:(NSString *)description
{
    self = [super init];

    if (self)
    {
        self.timeId = timeId;
        self.timeDescription = description;
        self.totalOtherEvents = 0;
    }

    return self;
}

/*  Decodes an existing object and returns the existing object with the encoded data    */
- (SBDayPartDescriptor *) initWithCoder:(NSCoder *) aDecoder
{
    // decode the encode objects
    NSString *description = (NSString *)[aDecoder decodeObjectForKey: @"timeDescription"];
    NSNumber *timeId = (NSNumber *)[aDecoder decodeObjectForKey: @"timeId"];
    NSDateComponents *start = (NSDateComponents *)[aDecoder decodeObjectForKey: @"startTime"];
    NSDateComponents *end = (NSDateComponents *)[aDecoder decodeObjectForKey: @"endTime"];

    // create the day part descriptor with the decoded information
    SBDayPartDescriptor *existingDescriptor = [[SBDayPartDescriptor alloc] initWithID: timeId andDescription:
        description];
    [existingDescriptor setStartTime: start];
    [existingDescriptor setEndTime: end];

    return existingDescriptor;
}

/*  Encodes the day part object */
- (void) encodeWithCoder:(NSCoder *) aCoder
{
    // encode the important information of the day part object
    [aCoder encodeObject: self.timeId forKey: @"timeId"];
    [aCoder encodeObject: self.timeDescription forKey: @"timeDescription"];
    [aCoder encodeObject: self.startTime forKey: @"startTime"];
    [aCoder encodeObject: self.endTime forKey: @"endTime"];
}

/*  Inverts the selected boolean value, returns the value that is currently being used after the invert  */
- (BOOL) toggleSelected
{
    _isSelected = !_isSelected;
    return _isSelected;
}

/*  Processes an event, checks wether the event occures in the time of this daypart, when true the event is added as
    a reference    */
- (void) processEvent:(EKEvent *)event withStartHour:(NSInteger) start endHour:(NSInteger) end
{
    // get the time start and end point
    NSInteger timeStart = [_startTime hour];
    NSInteger timeEnd = [_endTime hour];

    // check wether the event occurs on this daypart
    if ([event isAllDay] || (start >= timeStart && start <= timeEnd) || (end >= timeStart && end <= timeEnd) ||
        (start < timeStart && end > timeEnd))
    {
        // when no other events are scheduled already on this day part set the string description
        if (_totalOtherEvents == 0)
            _otherEventsDescription = [event title];

        // increase the event counter
        _totalOtherEvents++;
    }
}

/*  Overrider the string return method for the event others description */
- (NSString *) otherEventsDescription
{
    // when no events are plannend return a string indicating that
```

```
    if (_totalOtherEvents == 0)
        return @"No events planned.";

    // when more than one event is plannend return a formatted string
    if (_totalOtherEvents == 1)
        return _otherEventsDescription;
    else if (_totalOtherEvents == 2)
        return [NSString stringWithFormat: @"%@ & 1 other.", _otherEventsDescription];
    else return [NSString stringWithFormat: @"%@ & %d others.", _otherEventsDescription, _totalOtherEvents];
}

@end
```

```
//
//  SBDefectImageDescriptor.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
//#import "SBOperationManager.h"
//#import "SBImageThumbnailOperation.h"
//#import "SBImageWriteOperation.h"
#import "SBOperationDelegate.h"
#import "SBFileOperationDelegate.h"

typedef enum {
    SBImageLocationTypeCameraRoll           =   0,
    SBImageLocationTypeApplicationSandbox   =   1,
    SBImageLocationTypeInMemory             =   2
} SBImageLocationType;

#define SBImageThumbnailRect    256

@interface SBDefectImageDescriptor : NSObject <SBOperationDelegate, SBFileOperationDelegate>

// declare properties
@property (nonatomic, strong) NSURL *imageReferenceURL;
@property (nonatomic, strong) NSURL *metadataReferenceURL;
@property (nonatomic) SBImageLocationType imageLocationType;
@property (nonatomic, strong) NSDictionary *imageMetadata;

// when the image is in memory this image is stored
@property (nonatomic, strong) UIImage *inMemoryImage;

// the thumbnail of the image
@property (nonatomic, strong) UIImage *imageThumbnail;
@property (nonatomic, strong) NSURL *thumbnailURL;

// defines wether an image is currently being processed
@property (nonatomic) BOOL isProcessing;
@property (nonatomic, strong) NSError *processingError;

// creates a new image descriptor with the reference URL
- (SBDefectImageDescriptor *) initWithReferenceURL:(NSURL *) imageURL metaData:(NSDictionary *) metadata;
- (SBDefectImageDescriptor *) initWithImage:(UIImage *) image metadata:(NSDictionary *) metadata;

// start processing the images
- (void) startProcessing;
- (void) deleteCurrentImage;

@end
```

```
//
//  SBDefectImageDescriptor.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBDefectImageDescriptor.h"
#import <AssetsLibrary/AssetsLibrary.h>

#import "SBOperationManager.h"
#import "SBFileThumbnailOperation.h"
#import "SBImageFileSaveOperation.h"

// do some defines, for assets library to find URL's starting with it
static NSString *const ALAssetsLibaryScheme =   @"assets-library";

@implementation SBDefectImageDescriptor

/*  Initializes the image for a specific location type, can be camera roll or application sandbox   */
- (SBDefectImageDescriptor *) initWithReferenceURL:(NSURL *)imageURL metaData:(NSDictionary *)metadata
{
    self = [super init];

    if (self)
    {
        self.imageReferenceURL = imageURL;
        self.imageMetadata = metadata;
        self.isProcessing = NO;
    }

    return self;
}

/*  Creates a image descriptor object for in memory image   */
- (SBDefectImageDescriptor *) initWithImage:(UIImage *)image metadata:(NSDictionary *)metadata
{
    self = [super init];

    if (self)
    {
        self.inMemoryImage = image;
        self.imageMetadata = metadata;
        self.imageLocationType = SBImageLocationTypeInMemory;
        self.isProcessing = NO;
    }

    return self;
}

/*  Deletes the current saved image */
- (void) deleteCurrentImage
{
    // when the image reference URL is set and the scheme is not the asset library remove it from the documents
       folder
    if (self.imageReferenceURL != nil && ![[self.imageReferenceURL scheme] isEqualToString: ALAssetsLibaryScheme])
    {
        // create error object for reference
        NSError *error = nil;

        // create the file manager object that will remove the file
        NSFileManager *fileManager = [[NSFileManager alloc] init];

        // when successfull continue
        if (![fileManager removeItemAtURL: self.imageReferenceURL error: &error]){
            NSLog(@"Failed to remove the already saved file...");
        }

        // also remove the dictionary when possible
        if (self.metadataReferenceURL && ![fileManager removeItemAtURL: self.metadataReferenceURL error: &error]){
            NSLog(@"Failed to remove metadata reference file...");
        }
    }
}

/*  Starts the processing operation. always writes the thumbnail images to the application sandbox and when
    available also writes the original one to the user's camera roll */
- (void) startProcessing
{
    // when already processing return
    if (self.isProcessing){
        return;
    }

    // change processing value
    [self willChangeValueForKey: @"isProcessing"];
```

```
    self.isProcessing = YES;
    [self didChangeValueForKey: @"isProcessing"];

    // when the image is in memory, this means that a new image resource has been made by the camera
    if (self.inMemoryImage != nil) {

        // when authorized start the write to the saved photos album
        if ([ALAssetsLibrary authorizationStatus] == ALAuthorizationStatusAuthorized) {

            // start writing the image to the application sandbox
            ALAssetsLibrary *assetsLibrary = [[ALAssetsLibrary alloc] init];

            // start the write operation, we do not care whether the operation succeeded or not
            [assetsLibrary writeImageToSavedPhotosAlbum: [self.inMemoryImage CGImage] metadata: self.imageMetadata
                completionBlock: nil];
        }

        // create the operation that creates the thumbnail
        SBFileThumbnailOperation *thumbnailOperation = [[SBFileThumbnailOperation alloc] initWithImage: self.
            inMemoryImage options: nil];
        [thumbnailOperation setDelegate: self];

        // add the operation to the queue and start it
        [[SBOperationManager defaultManager] addOperation: thumbnailOperation];

    } else if (self.imageReferenceURL != nil) {

        // create the thumbnail operation
        SBFileThumbnailOperation *thumbnailOperation = [[SBFileThumbnailOperation alloc] initWithFileURL: self.
            imageReferenceURL withOptions: nil];
        [thumbnailOperation setDelegate: self];

        // start the operation in the queue
        [[SBOperationManager defaultManager] addOperation: thumbnailOperation];
    } else {

        // create user info for the error
        NSDictionary *userInfo = [NSDictionary dictionaryWithObject:
            NSLocalizedString(@"operation.image.process.failed.description", nil) forKey: NSLocalizedDescriptionKey]
            ;

        // no image found set the error object
        self.processingError = [NSError errorWithDomain: NSCocoaErrorDomain code: -1 userInfo: userInfo];

        // set processing to false
        [self willChangeValueForKey: @"isProcessing"];
        self.isProcessing = NO;
        [self didChangeValueForKey: @"isProcessing"];
    }
}

#pragma mark    -
#pragma mark    SBOperation delegate methods

/*  Gets called when the operation failed to execute    */
- (void) operation:(SBOperation *) operation didFailWithError:(NSError *) error
{
    // create user info for the error
    NSDictionary *userInfo = [NSDictionary dictionaryWithObject:
        NSLocalizedString(@"operation.image.process.failed.description", nil) forKey: NSLocalizedDescriptionKey];

    // no image found set the error object
    self.processingError = [NSError errorWithDomain: NSCocoaErrorDomain code: -1 userInfo: userInfo];

    // change value for the isProcessing key
    [self willChangeValueForKey: @"isProcessing"];
    self.isProcessing = NO;
    [self didChangeValueForKey: @"isProcessing"];
}

/*  Gets called when a thumbnail has been successfully created by the thumbnail operation   */
- (void) operation:(SBOperation *) operation didFinishProcessingImageWithResult:(UIImage *) image
{
    // change the thumbnail and isprocessing values
    self.imageThumbnail = image;

    // create and configure the write operation
    SBImageFileSaveOperation *saveOperation = [[SBImageFileSaveOperation alloc] initWithImage: self.imageThumbnail
        options: nil];
    [saveOperation setDelegate: self];

    // start the write operation
    [[SBOperationManager defaultManager] addOperation: saveOperation];
}

/*  Gets called when the operation has written its content to disk  */
- (void) operation:(SBOperation *) operation didFinishProcessingFileAtURL:(NSURL *) fileURL
```

```
{
    // set the new image
    self.thumbnailURL = fileURL;

    // free up memory by releasing the image
    self.inMemoryImage = nil;
    self.imageReferenceURL = nil;

    // change the image processing value
    self.isProcessing = NO;
    [self didChangeValueForKey: @"isProcessing"];
}

@end
```

```
//
//  SBFetchedResultsControllerAnimation.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 15-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface SBFetchedResultsControllerAnimation : NSObject

@property (nonatomic, strong) id changedObject;
@property (nonatomic) NSFetchedResultsChangeType changeType;
@property (nonatomic, strong) NSIndexPath *startingIndexPath;
@property (nonatomic, strong) NSIndexPath *endingIndexPath;

// create the object
- (SBFetchedResultsControllerAnimation *) initWithChangeType:(NSFetchedResultsChangeType) type object:(id) anObject
    startIndexPath:(NSIndexPath *) start endIndexPath:(NSIndexPath *) endIndexPath;

@end
```

```
//
//  SBFetchedResultsControllerAnimation.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 15-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBFetchedResultsControllerAnimation.h"

@implementation SBFetchedResultsControllerAnimation

/*  Initialization method of the fetched results controller animation   */
- (SBFetchedResultsControllerAnimation *) initWithChangeType:(NSFetchedResultsChangeType)type object:(id)anObject
    startIndexPath:(NSIndexPath *)start endIndexPath:(NSIndexPath *)endIndexPath
{
    self = [super init];

    if (self)
    {
        self.changeType = type;
        self.changedObject = anObject;
        self.startingIndexPath = start;
        self.endingIndexPath = endIndexPath;
    }

    return self;
}


@end
```

```
//
//  SBKeyboardToolbarLayout.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 19-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface SBKeyboardToolbarLayout : NSObject

// the different objects available for the toolbar, in this case a left & right bar button and a title
@property (nonatomic, strong) UIBarButtonItem *leftBarButtonItem;
@property (nonatomic, strong) UIBarButtonItem *rightBarButtonItem;
@property (nonatomic, strong) NSString *title;

// initializes the layout object with the provided bar buttons and title
+ (SBKeyboardToolbarLayout *) layoutWithLeftBarButtonItem:(UIBarButtonItem *) leftItem rightBarButtonItem:
    (UIBarButtonItem *) rightItem title:(NSString *) titleString;

// create a default layout object with no title and a next and previous button
+ (SBKeyboardToolbarLayout *) layoutWithLeftPreviousButtonRightNextButton;

@end
```

```
//
//  SBKeyboardToolbarLayout.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 19-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBKeyboardToolbarLayout.h"

@implementation SBKeyboardToolbarLayout

/*  Creates a toolbar layout object with the provided information buttons and title */
+ (SBKeyboardToolbarLayout *) layoutWithLeftBarButtonItem:(UIBarButtonItem *)leftItem rightBarButtonItem:
    (UIBarButtonItem *)rightItem title:(NSString *)titleString
{
    // create a new layout object
    SBKeyboardToolbarLayout *layout = [[SBKeyboardToolbarLayout alloc] init];

    // set properties
    [layout setLeftBarButtonItem: leftItem];
    [layout setRightBarButtonItem: rightItem];
    [layout setTitle: titleString];

    // return the created object
    return layout;
}

/*  Creates a toolbar layout with no title, but with the left bar button as a previous button and the right bar
    button as a next button */
+ (SBKeyboardToolbarLayout *) layoutWithLeftPreviousButtonRightNextButton
{
    // create a new layout object
    SBKeyboardToolbarLayout *layout = [[SBKeyboardToolbarLayout alloc] init];

    // create the default bar button items
    UIBarButtonItem *leftItem = [[UIBarButtonItem alloc] initWithTitle: NSLocalizedString(@"button.title.previous",
        nil) style: UIBarButtonItemStylePlain target: nil action: nil];
    UIBarButtonItem *rightItem = [[UIBarButtonItem alloc] initWithTitle: NSLocalizedString(@"button.title.next", nil
        ) style: UIBarButtonItemStylePlain target: nil action: nil];

    // set properties of the layout object
    [layout setLeftBarButtonItem: leftItem];
    [layout setRightBarButtonItem: rightItem];

    // return the layout object
    return layout;
}

@end
```

```
//
//  SBNavigationBarDetailView.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 16-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

// the default height of the navigation bar
#define kSBNavigationBarDetailViewHeight    50

// the max width of the detail description and title text
#define kSBNAvigationBarMaxTextWidth        250

@interface SBNavigationBarDetailView : UIView

// the image that is placed on top of all other views
@property (nonatomic, strong) UIImage *detailImage;

// text is placed on the bottom of the detail view container
@property (nonatomic, strong) NSString *detailText;

// an optional title can be passed to present between the views
@property (nonatomic, strong) NSString *detailTitle;
@property (nonatomic, strong) NSAttributedString *attributedTitleString;

// tint color that is used for the title detail text
@property (nonatomic, strong) UIColor *detailTintColor;

// initialize the object with the provided text
- (SBNavigationBarDetailView *) initWithDetailText:(NSString *) text;

// returns the calculated frame for this detail view that is minimally needed
- (CGSize) sizeThatFits:(CGSize) size;

@end
```

```
//
//  SBNavigationBarDetailView.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 16-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBNavigationBarDetailView.h"
#import "SBViewCompatabilityFactory.h"

@interface SBNavigationBarDetailView()

// creates the UILabel with description text for the UINAvigationBar
- (void) layoutSubviews;

// calcultes the appropriate frames for the different views, returns a CGSizeZero when not available
- (CGSize) sizeForDescription;
- (CGSize) sizeForTitle;
- (CGSize) sizeForImage;

// fonts used by the navigation bar
@property (nonatomic, strong) UIFont *titleFont;
@property (nonatomic, strong) UIFont *descriptionFont;

@end

@implementation SBNavigationBarDetailView

/*  Initializes the object with the provided text string  */
- (SBNavigationBarDetailView *) initWithDetailText:(NSString *) text
{
    // init super class
    self = [super init];

    if (self) {

        // set the detailed text and default tint color
        self.detailText = text;
        self.detailTintColor = [UIColor blackColor];

        // set the fonts used
        self.descriptionFont = [UIFont systemFontOfSize: 12];
        self.titleFont = [UIFont boldSystemFontOfSize: 13];
    }

    return self;
}

/*  Calculates the frames and places the subviews in the view container */
- (void) layoutSubviews
{
    CGSize descriptionSize = [self sizeForDescription];
    CGSize imageSize = [self sizeForImage];
    CGSize titleSize = [self sizeForTitle];

    // set the default y origin coordinate to start with
    CGFloat originY = 44;

    // when the image has a size the view can be added to the screen
    if (!CGSizeEqualToSize(imageSize, CGSizeZero)) {

        // create and set the frame of the new image view
        UIImageView *imageView = [[UIImageView alloc] initWithImage: self.detailImage];
        [imageView setFrame: CGRectMake((320 - imageSize.width) / 2, originY, imageSize.width, imageSize.height)];

        // add the image view to the superview
        [self addSubview: imageView];
        originY += (imageSize.height + 10);
    }

    // when the title is set create the label for the text
    if (!CGSizeEqualToSize(titleSize, CGSizeZero)) {

        // when the title is given create a label for this text
        UILabel *titleLabel = [[UILabel alloc] initWithFrame: CGRectMake((320 - titleSize.width) / 2, originY,
            titleSize.width, titleSize.height)];
        [titleLabel setFont: self.titleFont];

        // when the attributed text is set set this text to the label
        if (self.attributedTitleString != nil) {
            [titleLabel setAttributedText: self.attributedTitleString];
        } else {
            [titleLabel setText: self.detailTitle];
        }

        [titleLabel setTextAlignment: NSTextAlignmentCenter];
```

```
        [titleLabel setTextColor: self.detailTintColor];

        // add the view to the parent container and increase the origin
        [self addSubview: titleLabel];
        originY += titleSize.height;
    }

    // when the description text is given calculate the frame for that
    if (!CGSizeEqualToSize(descriptionSize, CGSizeZero)) {

        // calculate the origin to place in the center of the view
        CGFloat totalAvailableHeight = (kSBNavigationBarDetailViewHeight - (originY - 44));

        // create the label with the appropriate frame
        UILabel *descriptionLabel = [[UILabel alloc] initWithFrame: CGRectMake((320 - descriptionSize.width) / 2,
            originY + ((totalAvailableHeight - descriptionSize.height) / 2), descriptionSize.width, descriptionSize.
            height)];
        [descriptionLabel setFont: self.descriptionFont];
        [descriptionLabel setNumberOfLines: 2];
        [descriptionLabel setText: self.detailText];
        [descriptionLabel setTextAlignment: NSTextAlignmentCenter];

        // add the view to the parent container
        [self addSubview: descriptionLabel];
    }
}

/*  Return the size of the title that will be used  */
- (CGSize) sizeForTitle
{
    // check whether a title is set
    if (self.detailTitle != nil || self.attributedTitleString != nil) {

        // the default text that will be framed
        NSString *textToFrame = self.detailTitle;

        // when the attributed string is set the text to frame is the attributed text
        if (self.attributedTitleString != nil) {
            textToFrame = [self.attributedTitleString string];
        }

        // calculate the frame used for the title
        CGSize titleFrame = [SBViewCompatabilityFactory boundingRectForString: textToFrame withSize: CGSizeMake
            (kSBNAvigationBarMaxTextWidth, self.titleFont.lineHeight) options:
            NSStringDrawingTruncatesLastVisibleLine attributes: [NSDictionary dictionaryWithObject: self.titleFont
            forKey: NSFontAttributeName] context: nil];

        // return the frame size
        return titleFrame;

    } else {

        // return a zero size
        return CGSizeZero;
    }
}

/*  Calculates the size needed for the image view   */
- (CGSize) sizeForImage
{
    // check for an image to be available
    if (self.detailImage != nil) {

        // return the image size
        return [self.detailImage size];

    } else {

        // return zero because the view is unavailable
        return CGSizeZero;
    }
}

/*  Calculates the size of the description label thats minimum needed   */
- (CGSize) sizeForDescription
{
    // when no text is set this valu should not be calculated
    /*if (self.detailText != nil) {

        // calculate the minimum frame needed for the text
        CGSize descriptionSize = [SBViewCompatabilityFactory boundingRectForString: self.detailText withSize:
            CGSizeMake(kSBNAvigationBarMaxTextWidth, (self.descriptionFont.lineHeight * 2)) options:
            (NSStringDrawingUsesLineFragmentOrigin | NSStringDrawingUsesFontLeading) attributes: [NSDictionary
            dictionaryWithObject: self.descriptionFont forKey: NSFontAttributeName] context: nil];

        // return the calculated frame size
        return descriptionSize;
```

```
    } else {

        // return zero size to indicate no input
        return CGSizeZero;
    }*/

    return CGSizeZero;
}

/*  When the view will be added to its superview    */
- (void) willMoveToSuperview:(UIView *) newSuperview
{
    // start laying out the subviews
    [self layoutSubviews];
}

/*  Calculates and returns the size needed to fit in the provided size  */
- (CGSize) sizeThatFits:(CGSize) size
{
    // the default size of the navigation bar is 75 points
    return CGSizeMake(320, kSBNavigationBarDetailViewHeight);
}

@end
```

```
//
//  SBRelationDescriptor.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 15-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

// describes which relation
typedef enum
{
    SBRelationTypeDefectToElement       =   0,
    SBRelationTypeElementToSpace        =   1,
    SBRelationTypeSpaceToLocation       =   2
} SBRelationType;

@interface SBRelationDescriptor : NSObject

// descriptor properties
@property (nonatomic) SBRelationType relationType;
@property (nonatomic, retain) NSNumber *parentObjectID;
@property (nonatomic, retain) NSMutableArray *childObjectId;

// defines wether the parent object ID is available

// initializes an relation descriptor
- (SBRelationDescriptor *) initWithType:(SBRelationType) type parentId:(NSNumber *) identifier;
- (void) addChildObject:(NSManagedObjectID *) identifier;

@end
```

```
//
//  SBRelationDescriptor.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 15-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBRelationDescriptor.h"

@implementation SBRelationDescriptor

// initializes a relation descriptor
- (SBRelationDescriptor *) initWithType:(SBRelationType) type parentId:(NSNumber *) identifier
{
    self = [super init];

    if (self)
    {
        _relationType = type;
        _parentObjectID = identifier;
        _childObjectId = [[NSMutableArray alloc] initWithCapacity: 10];
    }

    return self;
}

- (void) addChildObject:(NSManagedObjectID *)identifier
{
    [_childObjectId addObject: identifier];
}

@end
```

```
//
//  SBReservationManager.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 26-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBOperationDelegate.h"
#import "SBURLOperationDelegate.h"
#import "SBReservationManagerDelegate.h"

@class Proceeding, Reservation, SBUser;

@interface SBReservationManager : NSObject <SBURLOperationDelegate, SBOperationDelegate,
    NSFetchedResultsControllerDelegate>

// the current reservation object that is being edited in the creation wizard
@property (nonatomic, strong) Reservation *currentReservation;
@property (nonatomic, weak) id<SBReservationManagerDelegate> activityDelegate;

// the current array index of the proceeding that is being edited
@property (nonatomic) NSInteger currentProceedingIndex;

// the fetched result controller for all reservation objects
@property (nonatomic, strong) NSFetchedResultsController *fetchedResultsController;

// staticly get the reservation manager
+ (SBReservationManager *) defaultReservationManager;

// processes the given value and key combination for the current proceeding
- (void) processValue:(id) value forCurrentProceedingPropertyKey:(NSString *) key;

// retreive an existing value for the given proceeding property key
- (id) valueForCurrentProceedingProperty:(NSString *) key;
- (SBUser *) userForCurrentReservation;

// when the view controller will start editing the current reservation object
- (void) willGenerateReservation;
- (void) didCancelReservation;

// call this method to complete the generation of the reservation and start the upload
- (void) completeReservationGenerating;
- (void) cancelUploadedReservation:(Reservation *) reservation;

// restarts the uploading process of a reservation
- (void) retryReservationUpload:(Reservation *) reservation;

// will start to synchronize all reservation objects with the ones in the API database
- (void) refreshReservationStatus;

// sets the new reservation fetched controller delegate object
- (void) setReservationControllerDelegate:(id<NSFetchedResultsControllerDelegate>) delegate;

@end
```

```
//
//  SBReservationManager.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 26-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBReservationManager.h"
#import "Reservation.h"
#import "Proceeding.h"
#import "SBAppDelegate.h"
#import "SBOperationManager.h"
#import "SBURLReservationOperation.h"
#import "SBAPIRequest.h"
#import "SBAPIResponse.h"

@interface SBReservationManager()

// the previous synchronization date for reservation objects
@property (nonatomic, strong) NSDate *lastSynchronizationDate;
@property (nonatomic) BOOL isSynchronizingReservations;

// the current upload counter
@property (nonatomic) NSUInteger currentUploadCounter;

// checks whether the reservation objects have states that need to be synchronized
- (BOOL) shouldSynchronize;

// parses the array of sync data
- (void) parseSynchronizationResult:(NSArray *) results;

// notifies the delegate when possible with the new activity description
- (void) notifyActivity;

@end

// static option names for operations
static NSString *SBReservationUploadOperationName   =   @"SBReservationUploadOperationName";
static NSString *SBReservationRefreshOperationName  =   @"SBReservationRefreshOperationName";
static NSString *SBReservationCancelOperationName   =   @"SBReservationCancelOperationName";

// define the minimum amount of seconds between sychronizations
#define kSBReservationSynchronizationtimeout    60

@implementation SBReservationManager

// the singleton object
static SBReservationManager *singleton = nil;

#pragma mark    Init & Singleton method

/*  the basic initializer method for the reservation manager    */
- (SBReservationManager *) init
{
    // call super initialization
    self = [super init];

    if (self) {

        // set default values
        self.isSynchronizingReservations = NO;
        self.currentUploadCounter = 0;

        // create the fetch request for reservation
        NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] initWithEntityName: @"Reservation"];
        [fetchRequest setSortDescriptors: [NSArray arrayWithObject: [NSSortDescriptor sortDescriptorWithKey:
            @"selectedDate" ascending: NO]]];

        // create the fetched results controller for reservations
        self.fetchedResultsController = [[NSFetchedResultsController alloc] initWithFetchRequest: fetchRequest
            managedObjectContext: [(SBAppDelegate *)[[UIApplication sharedApplication] delegate]
            managedObjectContext] sectionNameKeyPath: nil cacheName: nil];
        [self.fetchedResultsController setDelegate: self];

        // start to perform an initial fetch of all reservation objects
        NSError *error = nil;
        if (![self.fetchedResultsController performFetch: &error]) {
            NSLog(@"%@", error);
        }
    }

    return self;
}


/*  Get the shared manager object   */
```

```
+ (SBReservationManager *) defaultReservationManager
{
    // create the manager when not already existing
    if (singleton == nil) {
        singleton = [[SBReservationManager alloc] init];
    }

    return singleton;
}

/*  sets the new delegate of the fetched results controller and makes sure the delegate of this object is never nil
    */
- (void) setReservationControllerDelegate:(id<NSFetchedResultsControllerDelegate>)delegate
{
    // when the new delegate is nil set self as delegate to make sure notifications are still received
    if (delegate == nil) {
        [self.fetchedResultsController setDelegate: self];
    } else {
        [self.fetchedResultsController setDelegate: delegate];
    }
}

/*  Starts the refreshing of the current reservations in the fetched results controller */
- (void) refreshReservationStatus
{
    // when the last synchronization date is not set are more than 1 minute ago start the sync
    if ([self shouldSynchronize]) {

        // retrieve all objects from the controller
        NSArray *reservationObjects = [[[self.fetchedResultsController sections] objectAtIndex: 0] objects];

        // create string that will be used for post value
        NSMutableString *POSTString = [NSMutableString string];

        // loop through the objects and find the ones that needs syncing
        for (NSManagedObject *managedObject in reservationObjects) {

            // when the status is correct add to the POST values
            if ([[managedObject valueForKeyPath: @"currentStatus"] integerValue] == SBReservationStatusApproved ||
                [[managedObject valueForKeyPath: @"currentStatus"] integerValue] == SBReservationStatusWaiting) {

                // when a value is already added add a comma seperatoer
                if ([POSTString length] > 0) {
                    [POSTString appendString: @","];
                }

                // then append the string with the reservation external ID
                [POSTString appendFormat: @"%@", [managedObject valueForKeyPath: @"externalReservationId"]];
            }
        }

        // create the API request
        SBAPIRequest *syncRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPICheckStateRequestType timeout:
            60];
        [syncRequest setPOSTValue: POSTString forKey: @"reservation_list"];

        // create the operation object and configure it
        SBURLOperation *synchronizationOperation = [[SBURLOperation alloc] initWithAPIRequest: syncRequest options:
            @{SBOperationNameOptionKey: SBReservationRefreshOperationName}];
        [synchronizationOperation setDelegate: self];

        // add to the operation queue
        [[SBOperationManager defaultManager] addOperation: synchronizationOperation];
        self.isSynchronizingReservations = YES;

        // call delegate to inform activity
        [self notifyActivity];
    }
}

/*  Returns whether the application should start its synchronization task   */
- (BOOL) shouldSynchronize
{
    // check whether not currently synchronizing and whether the previous time
    if (!self.isSynchronizingReservations && (([self.lastSynchronizationDate timeIntervalSinceNow] >
        kSBReservationSynchronizationtimeout) || self.lastSynchronizationDate == nil)) {

        // retrieve all objects from the fetched results controller
        NSArray *allObjects = [[[self.fetchedResultsController sections] objectAtIndex: 0] objects];

        // enumerate all objects
        for (NSManagedObject *reservation in allObjects) {

            // when the state is waiting or approved the reservation should be synced
            if ([[reservation valueForKeyPath: @"currentStatus"] integerValue] == SBReservationStatusApproved ||
                [[reservation valueForKeyPath: @"currentStatus"] integerValue] == SBReservationStatusWaiting) {
```

```
                // when the first corresponding object is found the method should return YES
                return YES;
            }
        }
    }

    return NO;
}

/*  Parses the retrieved reservation synchronization results    */
- (void) parseSynchronizationResult:(NSArray *) results
{
    // get all managed objects in the fetched results controller
    NSArray *managedObjects = [[[self.fetchedResultsController sections] objectAtIndex: 0] objects];

    // enumerate through all dictionary array objects
    for (NSDictionary *syncResult in results) {

        // retrieve the external object ID and new state description
        NSNumber *externalReservationID = [syncResult objectForKey: @"id"];
        NSString *stateDescription = [syncResult objectForKey: @"state"];

        // find the reservation with the given external reservation id
        for (NSManagedObject *currentManagedObject in managedObjects) {

            // get the current status for checking for the same status as the previous one
            NSInteger currentObjectStatus = [[currentManagedObject valueForKeyPath: @"currentStatus"] integerValue];

            // when the reservation ID matches set the new state corresponding to the given string
            if ([[currentManagedObject valueForKeyPath: @"externalReservationId"] isEqualToNumber:
                externalReservationID]) {

                // set the status corresponding to the given state string
                if ([stateDescription isEqualToString: @"reserved"] && currentObjectStatus !=
                    SBReservationStatusWaiting) {
                    [currentManagedObject setValue: @(SBReservationStatusWaiting) forKeyPath: @"currentStatus"];
                } else if ([stateDescription isEqualToString: @"approved"] && currentObjectStatus !=
                    SBReservationStatusApproved) {
                    [currentManagedObject setValue: @(SBReservationStatusApproved) forKeyPath: @"currentStatus"];
                } else if ([stateDescription isEqualToString: @"denied"] && currentObjectStatus !=
                    SBReservationStatusDenied) {
                    [currentManagedObject setValue: @(SBReservationStatusDenied) forKeyPath: @"currentStatus"];
                } else if ([stateDescription isEqualToString: @"finished"] && currentObjectStatus !=
                    SBReservationStatusFinished) {
                    [currentManagedObject setValue: @(SBReservationStatusFinished) forKeyPath: @"currentStatus"];
                } else if ([stateDescription isEqualToString: @"cancelled"] && currentObjectStatus !=
                    SBReservationStatusCancelled) {

                    // delet the object from the context when cancelled
                    [[(SBAppDelegate *)[[UIApplication sharedApplication] delegate] managedObjectContext]
                        deleteObject: currentManagedObject];
                }
            }
        }
    }
}

/*  notifies the delegate object with a new activity description    */
- (void) notifyActivity
{
    // the default text string
    NSString *description = nil;

    // show text corresponding for upload and refresh
    if (self.isSynchronizingReservations && self.currentUploadCounter > 0) {
        description = NSLocalizedString(@"home.viewcontroller.load.description.both", nil);
    } else if (self.currentUploadCounter > 0) {
        description = NSLocalizedString(@"home.viewcontroller.load.description.upload", nil);
    } else if (self.isSynchronizingReservations) {
        description = NSLocalizedString(@"home.viewcontroller.load.description.refresh", nil);
    }

    // when the description is nil notify end of activity
    if (description == nil) {

        // notify the delegate that activity ended when possible
        if ([self.activityDelegate respondsToSelector: @selector(managerDidEndActivity:)]) {
            [self.activityDelegate managerDidEndActivity: self];
        }
    } else {

        // notify the delegate with new activity
        if ([self.activityDelegate respondsToSelector: @selector(manager:didUpdateActivityWithDescription:)]) {
            [self.activityDelegate manager: self didUpdateActivityWithDescription: description];
        }
    }
}
```

```
/*  Restarts the upload operation of a failed uploaded reservation  */
- (void) retryReservationUpload:(Reservation *)reservation
{
    // set the reservations status to uploading
    [reservation setCurrentStatus: [NSNumber numberWithInteger: SBReservationStatusUploading]];
    self.currentUploadCounter++;

    // create the API request for uploading the reservation
    SBURLReservationOperation *uploadOperation = [[SBURLReservationOperation alloc] initWithReservation: reservation
        options: @{SBOperationNameOptionKey: SBReservationUploadOperationName}];
    [uploadOperation setDelegate: self];

    // add the operation to the queue and notify the delegate
    [[SBOperationManager defaultManager] addOperation: uploadOperation];
    [self notifyActivity];
}

/*  Cancels the already uploaded reservation    */
- (void) cancelUploadedReservation:(Reservation *)reservation
{
    // create the API request for cancelling the reservation
    SBAPIRequest *cancelRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPICancelReservationRequestType
        timeout: 60];
    [cancelRequest setPOSTValue: [reservation externalReservationId] forKey: @"reservation_id"];

    SBURLOperation *cancelOperation = [[SBURLOperation alloc] initWithAPIRequest: cancelRequest options:
        @{SBOperationNameOptionKey: SBReservationCancelOperationName}];

    // set delegate and add to the queue
    [cancelOperation setDelegate: self];
    [[SBOperationManager defaultManager] addOperation: cancelOperation];
}

#pragma mark    -
#pragma mark    Managing the reservation info

/*  Set the new proceeding index and create a new proceeding when needed    */
- (void) setCurrentProceedingIndex:(NSInteger)currentProceedingIndex
{
    // set the current proceeding index
    NSInteger previousProceedingIndex = _currentProceedingIndex;
    _currentProceedingIndex = currentProceedingIndex;

    // check for the proceeding to exist already, when not create the proceeding
    if ([[self.currentReservation proceedings] count] == currentProceedingIndex) {

        // create the new proceeding entity managed object in the current managed object context
        NSManagedObjectContext *context = [(SBAppDelegate *)[[UIApplication sharedApplication] delegate]
            managedObjectContext];
        NSEntityDescription *proceedingEntity = [NSEntityDescription entityForName: @"Proceeding"
            inManagedObjectContext: context];
        Proceeding *newProceeding = [[Proceeding alloc] initWithEntity: proceedingEntity
            insertIntoManagedObjectContext: context];

        // add this proceeding object to the current reservation with the current index of the wizard
        [newProceeding setReservationProceedingIndex: [NSNumber numberWithInt: currentProceedingIndex]];
        [self.currentReservation addProceedingsObject: newProceeding];
    }

    // when the proceeding will be edited or created prepare its selected objects via this method
    NSEnumerator *setEnumerator = [[self.currentReservation proceedings] objectEnumerator];
    Proceeding *currentProceeding = nil;

    // find the current proceeding and call method that prepares the selection tree
    while (currentProceeding = [setEnumerator nextObject]) {
        if ([[currentProceeding reservationProceedingIndex] integerValue] == previousProceedingIndex) {
            [currentProceeding prepareSelectedDefectTreeForSelection: NO];
            break;
        } else if ([[currentProceeding reservationProceedingIndex] integerValue] == self.currentProceedingIndex) {
            [currentProceeding prepareSelectedDefectTreeForSelection: YES];
            break;
        }
    }
}

/*  Retrieve an existing value from the proceeding object currently being edited    */
- (id) valueForCurrentProceedingProperty:(NSString *)key
{
    // create the object enumerator
    NSEnumerator *setEnumerator = [[self.currentReservation proceedings] objectEnumerator];
    Proceeding *currentProceeding = nil;

    // loop through the set to find the proceeding at the current index
    while (currentProceeding = [setEnumerator nextObject]) {

        // when the indexes are equal return the objects value for the given key
```

```
        if ([[currentProceeding reservationProceedingIndex] integerValue] == self.currentProceedingIndex) {
            return [currentProceeding valueForKey: key];
        }
    }

    return nil;
}

/*  When a new reservation will be started to create this method gets called    */
- (void) willGenerateReservation
{
    // when the reservation is not yet set, create a new reservation object
    if (self.currentReservation == nil) {

        // get the managed object contexta and entity description needed for a new reservation managed object
        NSManagedObjectContext *context = [(SBAppDelegate *)[[UIApplication sharedApplication] delegate]
            managedObjectContext];
        NSEntityDescription *entityDescription = [NSEntityDescription entityForName: @"Reservation"
            inManagedObjectContext: context];

        // set the new reservation managed object
        self.currentReservation = [[Reservation alloc] initWithEntity: entityDescription
            insertIntoManagedObjectContext: context];
    }
}

/*  Cancels the current reservation that is being created, remove from core data   */
- (void) didCancelReservation
{
    // when a current reservation is available remove it from context
    if (self.currentReservation != nil) {

        // remove from the managed object context
        NSManagedObjectContext *context = [(SBAppDelegate *)[[UIApplication sharedApplication] delegate]
            managedObjectContext];
        [context deleteObject: self.currentReservation];

        // reset the current reservation
        self.currentReservation = nil;
    }
}

/*  when the wizrad has successfully created the new reservation and this reservation will start to be uploaded */
- (void) completeReservationGenerating
{
    // start the upload process of the generated reservation
    if (self.currentReservation != nil) {

        // set the reservations status to uploading
        [self.currentReservation setCurrentStatus: [NSNumber numberWithInteger: SBReservationStatusUploading]];
        self.currentUploadCounter++;

        // create the API request for uploading the reservation
        SBURLReservationOperation *uploadOperation = [[SBURLReservationOperation alloc] initWithReservation: self.
            currentReservation options: @{SBOperationNameOptionKey: SBReservationUploadOperationName}];
        [uploadOperation setDelegate: self];

        // add the operation to the queue and notify the delegate
        [[SBOperationManager defaultManager] addOperation: uploadOperation];
        [self notifyActivity];
    }

    // reset the current reservation
    self.currentReservation = nil;
}

/*  Process a new value to the current proceeding object    */
- (void) processValue:(id) value forCurrentProceedingPropertyKey:(NSString *) key
{
    // override the date and daypart, this will be saved in the reservation object instead of the proceeding at the
        current index
    if ([key isEqualToString: @"reservationDate"]) {
        [self.currentReservation setSelectedDate: value];
    }
    else if ([key isEqualToString: @"reservationDayPart"]) {
        [self.currentReservation setDaypart: value];
    }
    else {

        // create the object enumerator
        NSEnumerator *setEnumerator = [[self.currentReservation proceedings] objectEnumerator];
        Proceeding *currentProceeding = nil;

        // loop through the set to find the proceeding at the current index
        while (currentProceeding = [setEnumerator nextObject]) {

            // when the indexes are equal set the given value and key for this proceeding object
```

```
            if ([[currentProceeding reservationProceedingIndex] integerValue] == self.currentProceedingIndex) {
                [currentProceeding setValue: value forKey: key];
            }
        }
    }
}

/*  Return the user object for the current reservation being edited */
- (SBUser *) userForCurrentReservation
{
    // return the user
    return [self.currentReservation reservationUser];
}

#pragma mark    -
#pragma mark    SBOperation delegate

/*  When an operation failed to execute with the provided error this method is called   */
- (void) operation:(SBOperation *) operation didFailWithError:(NSError *) error
{
    // when the operation name is an upload operation check the result
    if ([[[operation options] valueForKey: SBOperationNameOptionKey] isEqualToString:
        SBReservationUploadOperationName]) {

        // lower current counter and notify activity
        self.currentUploadCounter--;

    } else if ([[[operation options] valueForKey: SBOperationNameOptionKey] isEqualToString:
        SBReservationRefreshOperationName]) {

        // notify delegate with activity change
        self.isSynchronizingReservations = NO;
    }

    // notify delegate
    [self notifyActivity];
}

/*  When an operation succeeded its execution this method is called with the appropriate response   */
- (void) operation:(SBOperation *)operation didFinishWithResult:(SBAPIResponse *)response
{
    // when the operation name is an upload operation check the result
    if ([[[operation options] valueForKey: SBOperationNameOptionKey] isEqualToString:
        SBReservationUploadOperationName]) {

        // cast the operation as upload operation
        SBURLReservationOperation *executedOperation = (SBURLReservationOperation *)operation;

        // an upload operation finished, check whether successfull
        if ([response objectResultType] == SBAPIResultSuccessType) {

            // set the new state of the reservation object
            [[executedOperation reservation] setCurrentStatus: [NSNumber numberWithInteger:
                SBReservationStatusWaiting]];
            [[executedOperation reservation] setIsUploaded: [NSNumber numberWithBool: YES]];
        } else {

            // set the current status to failed
            [[executedOperation reservation] setCurrentStatus: [NSNumber numberWithInteger:
                SBReservationStatusFailed]];
        }

        // lower current counter and notify activity
        self.currentUploadCounter--;

    } else if ([[[operation options] valueForKey: SBOperationNameOptionKey] isEqualToString:
        SBReservationRefreshOperationName]) {

        // when successfully synchronized set the last sync date and parse the data
        if ([response objectResultType] == SBAPIResultSuccessType) {

            // set the new synchronization date
            self.lastSynchronizationDate = [NSDate date];
            [self parseSynchronizationResult: response.resultDictionary];
        }

        // notify delegate with activity change
        self.isSynchronizingReservations = NO;
    } else if ([[[operation options] valueForKey: SBOperationNameOptionKey] isEqualToString:
        SBReservationCancelOperationName]) {

        // when successfully cancelled append the state of the reservation
        if ([response objectResultType] == SBAPIResultSuccessType) {

            // get the external id of the managed object that was cancelled
            NSNumberFormatter *numberFormatter = [[NSNumberFormatter alloc] init];
            NSNumber *externalReservationId = [numberFormatter numberFromString: [[response resultDictionary]
```

```
                objectForKey: @"id"]];
            NSArray *managedObjects = [[[self.fetchedResultsController sections] objectAtIndex: 0] objects];

            // enumarate all reservation objects
            for (NSManagedObject *currentReservation in managedObjects) {

                // when the external reservation id is equal set cancelled and delete from context
                if ([[currentReservation valueForKeyPath: @"externalReservationId"] isEqualToNumber:
                    externalReservationId]) {

                    // delete the object from the context and break
                    [[(SBAppDelegate *) [[UIApplication sharedApplication] delegate] managedObjectContext]
                        deleteObject: currentReservation];
                    break;
                }
            }
        } else {

            // create an alert view to indicate that the cancelling process has failed
            UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
                NSLocalizedString(@"overview.viewcontroller.cancel.failed.title", nil) message:
                NSLocalizedString(@"overview.viewcontroller.cancel.failed.description", nil) delegate: nil
                cancelButtonTitle: nil otherButtonTitles: NSLocalizedString(@"alertview.button.title.dismiss", nil),
                nil];

            // show the alert
            [alertView show];
        }
    }

    // notify the delegate
    [self notifyActivity];
}

#pragma mark    -
#pragma mark    NSFetchedResultsController delegate

/*  Implement all fetched results controller delegate methods to make sure the fetched results controller stays up
    to date about managed object context changes and its reservation objects */
- (void) controllerDidChangeContent:(NSFetchedResultsController *) controller{}

@end
```

```
//
//  SBUser.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 18-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBCustomerDescriptor.h"

// results key for validation
static NSString * const SBUserValidationResultKeyZipCode        =   @"ValidatedZipCode";
static NSString * const SBUserValidationResultKeyHouseNumber    =   @"ValidatedHouseNumber";
static NSString * const SBUserValidationResultKeyAddition       =   @"ValidatedAddition";
static NSString * const SBUserValidationResultKeyMergedNumber   =   @"ValidatedMergedNumber";

// nsuser defaults keys
static NSString * const SBUserDefaultsKeyName               =   @"user.name";
static NSString * const SBUserDefaultsKeyEmail              =   @"user.email";
static NSString * const SBUserDefaultsKeyPhone              =   @"user.phone.default";
static NSString * const SBUserDefaultsKeyPhoneOptional      =   @"user.phone.optional";
static NSString * const SBUserDefaultsKeyZipcode            =   @"user.zipcode";
static NSString * const SBUserDefaultsKeyHouseNumber        =   @"user.house_number";
static NSString * const SBUserDefaultsDeviceToken           =   @"device.token";
static NSString * const SBUserDefaultsKeyStreet             =   @"user.address.street";
static NSString * const SBUserDefaultsKeyCity               =   @"user.address.city";
static NSString * const SBUserDefaultsKeyCustomer           =   @"user.customer";

@interface SBUser : NSObject

// the properties of the user, name and adress info
@property (nonatomic, strong) NSString *userFirstName;
@property (nonatomic, strong) NSString *userEmail;
@property (nonatomic, strong) NSString *userTelephoneNumber;
@property (nonatomic, strong) NSString *userOptionalTelephoneNumber;
@property (nonatomic, strong) NSString *deviceToken;

// address information for the user
@property (nonatomic, strong) NSString *zipcode;
@property (nonatomic, strong) NSString *houseNumber;
@property (nonatomic, strong) NSString *addition;
@property (nonatomic, strong) NSString *street;
@property (nonatomic, strong) NSString *town;

// the customer from the user
@property (nonatomic, strong) SBCustomerDescriptor *userCustomer;

// decares wether the contents culd be fetched from the user defaults
@property (nonatomic) BOOL isFetchedFromUserDefaults;

// when called the data will be tried to fetch from userdefaults
- (BOOL) fetchFromUserDefaults;
+ (BOOL) isValidEmail:(NSString *) emailAddress;
+ (BOOL) validateString:(NSString *) string withAllowedCharacterSet:(NSCharacterSet *) allowedCharacters minLength:
    (NSUInteger)minLength maxLength:(NSUInteger)maxLength;
+ (BOOL) validatePhoneNumber:(NSString *) phoneNumber;

// validates the string for normal characters used for a name
+ (BOOL) validateFullNameString:(NSString *) string;

// retrieve the shared user
+ (SBUser *) sharedUser;

// returns the data string for POST uploading
- (void) unsetAddress;
- (BOOL) isValidated;
- (void) save;

// returns wether the given zipcode and house number is valid form
+ (BOOL) isValidZipCode:(NSString *) zipcode withHouseNumber:(NSString *) number results:(NSDictionary **) results;

@end
```

```
//
//  SBUser.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 18-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBUser.h"

@implementation SBUser

// the static user used in the application
static SBUser *userSingleton = nil;

/*  Returns wether the user could be loaded from the userdefaults   */
- (BOOL) fetchFromUserDefaults
{
    // get the user defaults object
    NSUserDefaults *userDefaults = [NSUserDefaults standardUserDefaults];

    // retreive and set all user defaults values
    self.userFirstName = [userDefaults valueForKey: SBUserDefaultsKeyName];
    self.userEmail = [userDefaults valueForKey: SBUserDefaultsKeyEmail];
    self.userTelephoneNumber = [userDefaults valueForKey: SBUserDefaultsKeyPhone];
    self.userOptionalTelephoneNumber = [userDefaults valueForKey: SBUserDefaultsKeyPhoneOptional];
    self.zipcode = [userDefaults valueForKey: SBUserDefaultsKeyZipcode];
    self.houseNumber = [userDefaults valueForKey: SBUserDefaultsKeyHouseNumber];
    self.deviceToken = [userDefaults valueForKey: SBUserDefaultsDeviceToken];
    self.street = [userDefaults valueForKey: SBUserDefaultsKeyStreet];
    self.town = [userDefaults valueForKey: SBUserDefaultsKeyCity];
    self.userCustomer = [userDefaults valueForKey: SBUserDefaultsKeyCustomer];

    // fetche dictionary from the user dfeaults
    NSDictionary *customerDictionary = [userDefaults valueForKey: SBUserDefaultsKeyCustomer];

    // when the name value could be fetched notify that user defaults have been fetched from disk
    if ([self.userFirstName length] > 0) {

        // parse the customer and set the current customer
        SBCustomerDescriptor *descriptor = [[SBCustomerDescriptor alloc] init];
        [descriptor setCustomerId: [customerDictionary objectForKey: @"customerId"]];
        [descriptor setCustomerName: [customerDictionary objectForKey: @"customerName"]];
        self.userCustomer = descriptor;

        return YES;
    }
    else {
        return NO;
    }
}

/*  Save the user data to the user defaults */
- (void) save
{
    // get the standard user defaults for the application
    NSUserDefaults *userDefaults = [NSUserDefaults standardUserDefaults];

    // set all user defaults values
    [userDefaults setObject: self.userFirstName forKey: SBUserDefaultsKeyName];
    [userDefaults setObject: self.userEmail forKey: SBUserDefaultsKeyEmail];
    [userDefaults setObject: self.userTelephoneNumber forKey: SBUserDefaultsKeyPhone];
    [userDefaults setObject: self.userOptionalTelephoneNumber forKey: SBUserDefaultsKeyPhoneOptional];
    [userDefaults setObject: self.zipcode forKey: SBUserDefaultsKeyZipcode];
    [userDefaults setObject: self.houseNumber forKey: SBUserDefaultsKeyHouseNumber];
    [userDefaults setObject: self.street forKey: SBUserDefaultsKeyStreet];
    [userDefaults setObject: self.town forKey: SBUserDefaultsKeyCity];

    // fetch the customer description from the user defaults
    NSDictionary *customerDescripiton = [NSDictionary dictionaryWithObjectsAndKeys: self.userCustomer.customerId,
        @"customerId", self.userCustomer.customerName, @"customerName", nil];
    [userDefaults setObject: customerDescripiton forKey: SBUserDefaultsKeyCustomer];
}

/*  Returns the shared user object  */
+ (SBUser *) sharedUser
{
    // when the singleton is not yet initialized create it
    if (userSingleton == nil) {
        userSingleton = [[SBUser alloc] init];
        [userSingleton fetchFromUserDefaults];
    }

    // return the singleton
    return userSingleton;
}
```

```
/*  Unsets all address values   */
- (void) unsetAddress
{
    // reset all address values
    self.street = nil;
    self.town = nil;
    self.zipcode = nil;
    self.houseNumber = nil;
    self.addition = nil;
}

/*  Returns wether all values of the object have been set and thus been validated   */
- (BOOL) isValidated
{
    // when an object is nil return NO
    if (self.userFirstName == nil || self.userEmail == nil || self.userTelephoneNumber == nil || self.street == nil
        || self.userCustomer == nil) {
        return NO;
    }
    else {
        return YES;
    }
}

#pragma mark    -
#pragma mark    Validation static methods

/*  Returns wether the passed email address is valid    */
+ (BOOL) isValidEmail:(NSString *) emailAddress
{
    // this is the email validation regular expression from RFC 2822, only allowews .web domains which should be
        invalid and IP validation is not perfect.
    NSString *emailRegex = @"(?:[a-z0-9!#$%\\&'*+/=?\\^_`{|}~-]+(?:\\.[a-z0-9!#$%\\&'*+/=?\\^_`{|}~-]+)*|\"(?:[\
        \x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])*\")@(?:
        (?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?
        [0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\\x01-\\x08\\x0b\\x0c\
        \x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])+)\\])";

    // validate the email address using the above regular expression and this predicate
    NSPredicate *emailPredicate = [NSPredicate predicateWithFormat:  @"SELF MATCHES[c] %@", emailRegex];
    return [emailPredicate evaluateWithObject: emailAddress];
}

/*  Validates the string with the provided characterset, when the character set is null a default character set will
    be used. The string must be between the provided stirng lengths */
+ (BOOL) validateString:(NSString *) string withAllowedCharacterSet:(NSCharacterSet *) allowedCharacters minLength:
    (NSUInteger)minLength maxLength:(NSUInteger)maxLength
{
    // get the string length
    NSInteger stringLength = [string length];

    // when the string length is between the length continue
    if (stringLength >= minLength && stringLength <= maxLength)
    {
        // validate the string with the inverted charater set, when the characters are not found this string is
            valid
        if ([string rangeOfCharacterFromSet: [allowedCharacters invertedSet]].location != NSNotFound){
            return NO;
        }
        else return YES;
    }
    else return NO;
}

/*  Validates the given phone number, can be a dutch 06 mobile number or normal net code. Returns the new string
    without junk  */
+ (BOOL) validatePhoneNumber:(NSString *) phoneNumber
{
    // create the scanner object
    NSScanner *phoneScanner = [NSScanner scannerWithString: phoneNumber];
    NSCharacterSet *validCharacters = [NSCharacterSet characterSetWithCharactersInString: @"0123456789"];
    NSMutableString *strippedPhoneNumber = [[NSMutableString alloc] init];
    NSString *tempStripped = nil;

    // scan through the string and skip all characters except the numeric ones
    while (![phoneScanner isAtEnd]) {

        // scan characters until invalid
        if ([phoneScanner scanCharactersFromSet: validCharacters intoString: &tempStripped]) {
            [strippedPhoneNumber appendString: tempStripped];
        }

        // set the current scan location
        if ([phoneScanner scanLocation] < [phoneNumber length]) {
            [phoneScanner setScanLocation: ([phoneScanner scanLocation] + 1)];
        }
    }
```

```
    // when larger than 10 characters strip the first occurences of zero
    if ([strippedPhoneNumber length] > 10) {

        // keep replacing the character at the first index untill it is not a 0 anymore
        while ([strippedPhoneNumber characterAtIndex: 0] == '0') {
            [strippedPhoneNumber replaceCharactersInRange: NSMakeRange(0, 1) withString: @""];
        }
    }

    // create the telehone number regex specialized for dutch phone numbers
    NSString *telephoneRegex = @"^(31|310|0){1}(([6]{1}[1-9]{1}[0-9]{7})|([12345789]{1}[0-9]{8})|([8-9]{1}[0]{2}
        [0-9]{7}))$";

    // create the regular expression object
    NSError *error = nil;
    NSRegularExpression *regularExpression = [NSRegularExpression regularExpressionWithPattern: telephoneRegex
        options: NSRegularExpressionCaseInsensitive error: &error];

    // when matches have been found the phone number is valid
    if ([regularExpression numberOfMatchesInString: strippedPhoneNumber options: 0 range: NSMakeRange(0,
        strippedPhoneNumber.length)] > 0) {
        return YES;
    } else {
        return NO;
    }
}

/*  Returns wether the zipcode and house number are valid in the form   */
+ (BOOL) isValidZipCode:(NSString *)zipcode withHouseNumber:(NSString *)number results:(NSDictionary **) results
{
    // create a zipcode validater
    NSString *zipcodeRegex = @"^[1-9]{1}[0-9]{3} ?[a-zA-Z]{2}$";

    // when no number is given, return NO immediatly
    if ([number length] <= 0 && [number length] <= 15) {
        return NO;
    }

    // scan the house number
    NSScanner *numberScanner = [NSScanner scannerWithString: number];
    NSCharacterSet *characterSet = [NSCharacterSet characterSetWithCharactersInString: @"0123456789"];
    NSMutableString *parsedString = [NSMutableString stringWithCapacity: [number length]];

    // while the scanner is not at the end continue
    while (![numberScanner isAtEnd]){
        NSString *stringBuffer;

        // when characters are available from the characterset append the target string
        if ([numberScanner scanCharactersFromSet: characterSet intoString: &stringBuffer]){
            [parsedString appendString: stringBuffer];
        }
        else{
            // set the location to end, because the numbers are expected at the start of the string
            [numberScanner setScanLocation: [number length]];
        }
    }

    // when the parsed string length is bigger or equal than 6 (max int of 99999)
    if ([parsedString length] >= 6) {
        return NO;
    }

    // create the predicate to validate the zipcode
    NSPredicate *zipcodePredicate = [NSPredicate predicateWithFormat: @"SELF MATCHES[c] %@", zipcodeRegex];

    // validate wether the house number has 1 or more characters and the zipcode is of valid form
    if ([zipcodePredicate evaluateWithObject: zipcode] && [parsedString length] > 0){
        // create the addition string
        NSString *houseAddition = [NSString stringWithString: [[number substringFromIndex: [parsedString length]]
            stringByTrimmingCharactersInSet: [NSCharacterSet whitespaceAndNewlineCharacterSet]]];

        NSString *mergedNumber = nil;

        // create a space between the nunber and addition
        if ([houseAddition length] > 0){
            mergedNumber = [NSString stringWithFormat: @"%@ %@", parsedString, houseAddition];
        }
        else{
            mergedNumber = parsedString;
        }

        // set the reference pointer dictionary
        *results = [NSDictionary dictionaryWithObjectsAndKeys: [[zipcode uppercaseString]
            stringByTrimmingCharactersInSet: [NSCharacterSet whitespaceAndNewlineCharacterSet]],
            SBUserValidationResultKeyZipCode, parsedString, SBUserValidationResultKeyHouseNumber, houseAddition,
            SBUserValidationResultKeyAddition, mergedNumber, SBUserValidationResultKeyMergedNumber, nil];
```

```
        // validation succeeded, return YES
        return YES;
    }
    else return NO;
}

/*  Validates whether the given name (First and lastname is valid) contains valid characters    */
+ (BOOL) validateFullNameString:(NSString *) string
{
    // create the character set with all characters that are allowed for the full name
    NSCharacterSet *invalidCharacterSet = [NSCharacterSet characterSetWithCharactersInString: @"0123456789"];
    BOOL returnValue = NO;

    // the full name must be at least 2 characters long and can be a maximum of 150 characters long
    if ([string length] >= 2 && [string length] <= 150) {

        // when the string only contains characters from the provided set the return value will be YES, else the
            value will become NO
        returnValue = ([string rangeOfCharacterFromSet: invalidCharacterSet options: NSCaseInsensitiveSearch].
            location == NSNotFound);
    }

    // return the validation result
    return returnValue;
}

@end
```

```
//
//  SBVerificationManager.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 06-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBOperationDelegate.h"
#import "SBURLOperationDelegate.h"

// the user defaults identifier for the username used in basic HTTP authentication
static NSString *SBVerificationUsernameKey      =   @"verification.username";
static NSString *SBVerificationTokenKey         =   @"verification.token";
static NSString *SBVerificationUUIDKey          =   @"verification.uuid";

// the notification name that is emitted when new token and key is fetched
static NSString *SBVerificationManagerDidRefreshAuthenticationTokenNotification =   @"AuthenticationTokenDidRefresh"
    ;

@interface SBVerificationManager : NSObject <SBOperationDelegate, SBURLOperationDelegate, UIAlertViewDelegate>
{
    // whether the connection to the API server is already open
    BOOL isRefreshingToken;
}

// the username used for the secure connection, this must be first fetched from the API
@property (nonatomic, strong) NSString *secureConnectionUsername;
@property (nonatomic, strong) NSString *secureConnectionToken;
@property (nonatomic, strong) NSString *secureDeviceUUID;

// the array with a list of all domains which can be connected with
@property (nonatomic, strong) NSArray *allowedDomains;

//the base 64 authorization string
@property (nonatomic, strong) NSString *base64Authorization;

// retrieves the statis singleton of the verification manager
+ (SBVerificationManager *) defaultVerification;

// returns whether the domain protection space is valid to use for this application
- (BOOL) applicationIsAllowedToConnectUsingProtectionSpace:(NSURLProtectionSpace *) protectionSpace;

// changes the given authentintication to use the specified credentials
- (void) processConnectionWithAuthenticationChallenge:(NSURLAuthenticationChallenge *) authenticationChallenge;

// validate username and password token
- (BOOL) hasValidVerificationCredentials;
- (void) saveCredentials;

// generates the base 64 string
- (NSString *) generateBase64AuthorizationString;


@end
```

```
//
//  SBVerificationManager.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 06-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBVerificationManager.h"
#import "SBURLOperation.h"
#import "SBAPIRequest.h"
#import "SBAPIResponse.h"
#import "SBOperationManager.h"
#import "SBViewCompatabilityFactory.h"
#include <resolv.h>

@interface SBVerificationManager()

@end

@implementation SBVerificationManager

// the singleton object used by the application to verify and validate server protection space etc.
static SBVerificationManager *sharedVerificationManager = nil;

/*  Initializer method which retrieves the unique identifier used for passwords to connect to the server    */
- (SBVerificationManager *) init
{
    // create parent object
    self = [super init];

    if (self) {

        // fetch the user defaults username object
        NSUserDefaults *userDefaults = [NSUserDefaults standardUserDefaults];
        id usernameObject = [userDefaults objectForKey: SBVerificationUsernameKey];
        id tokenObject = [userDefaults objectForKey: SBVerificationTokenKey];
        id UUID = [userDefaults objectForKey: SBVerificationUUIDKey];

        // when the username object fetched from the defaults is not nil and is a string object set the current
            username to be used for basic authentication
        if (usernameObject != nil && [usernameObject isKindOfClass: [NSString class]]) {
            self.secureConnectionUsername = (NSString *)usernameObject;
        }

        // find the token
        if (tokenObject != nil && [tokenObject isKindOfClass: [NSString class]]) {
            self.secureConnectionToken = (NSString *)tokenObject;
        }

        // find the specific device token
        if (UUID != nil && [UUID isKindOfClass: [NSString class]]) {
            self.secureDeviceUUID = (NSString *)UUID;
        }

        // set the allowed domains that can be connected to by the application
        self.allowedDomains = [NSArray arrayWithObjects: @"stephan.bc-dev.dev.solware.local", @"bc-test.solware.nl",
            nil];
        isRefreshingToken = NO;
    }

    // return the verification manager object, this has a valid identifier
    return self;
}

/*  Return the default verification manager that should be used */
+ (SBVerificationManager *) defaultVerification
{
    // when the singleton is not yet created try to create the object
    if (sharedVerificationManager == nil) {
        sharedVerificationManager = [[SBVerificationManager alloc] init];
    }

    // return the created manager, this value can be nil when the identifier could not be set
    return sharedVerificationManager;
}

/*  Returns whether the application can connect to the server which provided the protectin space    */
- (BOOL) applicationIsAllowedToConnectUsingProtectionSpace:(NSURLProtectionSpace *) protectionSpace
{
    // set the return value
    BOOL isValidProtectionSpace = NO;

    // check whether the SSL port is being used for the connection
    /*if ([protectionSpace port] != 443 || ![[protectionSpace protocol] isEqualToString: @"https"]) {
        return isValidProtectionSpace;
    }*/
```

```
    // create object enumerator to validate the domain
    NSEnumerator *domainEnumerator = [self.allowedDomains objectEnumerator];
    NSString *currentDomain = nil;

    // loop through all domains and check whether the given domain matches one of them
    while (currentDomain = [domainEnumerator nextObject]) {

        // when the domain matches set
        if ([currentDomain isEqualToString: [protectionSpace host]]) {
            isValidProtectionSpace = YES;
            break;
        }
    }

    // successfully validated the protection space, return the result
    return isValidProtectionSpace;
}

/*  Process the authentication challenge and sets the proper credentials when the connection can be accepted, in
    this method the certificate will be verified for server trust  */
- (void) processConnectionWithAuthenticationChallenge:(NSURLAuthenticationChallenge *) authenticationChallenge
{
    // when multiple authentication errors have occured immediatly cancel
    if ([authenticationChallenge previousFailureCount] > 0) {
        [[authenticationChallenge sender] cancelAuthenticationChallenge: authenticationChallenge];
        return;
    }

    // when server trust has been validated we are expected to validate via basic HTTP authentication, fetch the
        credentials and use this for the authentication challenge
    if ([[[authenticationChallenge protectionSpace] authenticationMethod] isEqualToString:
        NSURLAuthenticationMethodDefault]) {

        // create the new credential to be used for this connection
        NSURLCredential *secureConnectionCredential = [NSURLCredential credentialWithUser: self.
            secureConnectionUsername password: self.secureConnectionToken persistence:
            NSURLCredentialPersistenceNone];

        // set that the challenge should use the created credentials
        [[authenticationChallenge sender] useCredential: secureConnectionCredential forAuthenticationChallenge:
            authenticationChallenge];
    } else if ([[[authenticationChallenge protectionSpace] authenticationMethod] isEqualToString:
        NSURLAuthenticationMethodServerTrust]) {

        // create the array of certificates to be validated with the amount of sertificaes in the authentication
            trust
        CFIndex certificateCount = SecTrustGetCertificateCount(authenticationChallenge.protectionSpace.serverTrust);
        SecCertificateRef certificates[certificateCount];

        // add each available certificate to the array, the first certificate in the array will be the outer leaf
            certificate. In our case the solware certificate followed by root certificates.
        for (CFIndex i = 0; i < certificateCount; i++) {
            certificates[i] = SecTrustGetCertificateAtIndex(authenticationChallenge.protectionSpace.serverTrust, i);
        }

        // create the used certificate array reference to use for creating our custom trust reference
        CFArrayRef certificateArray = CFArrayCreate(kCFAllocatorDefault, (void *)certificates, certificateCount,
            NULL);

        // create a new policy to be trusted, in our case this policy is the solware api server. Do not trust any
            other servers because we know that the only point of communication is this host. Use * for all sub
            domains
        SecPolicyRef trustedServer = SecPolicyCreateSSL(true, (CFStringRef)@"*.solware.nl");

        // create a new trust reference with the provided custom policy and certificates provided in the
            authentication chalenge. The policy will make sure validation will happen for solware hosts
        SecTrustRef newTrustedCertificate = NULL;
        OSStatus verificationResult = SecTrustCreateWithCertificates(certificateArray, trustedServer, &
            newTrustedCertificate);

        // validate for the creation to be successfull
        if (verificationResult == errSecSuccess) {

            // now evaluate the certificate of the server and save the result type
            SecTrustResultType verificationResultType;
            OSStatus serverVerificationResult = SecTrustEvaluate(newTrustedCertificate, &verificationResultType);

            // when the os status is successfull and the the verification result is proceed or unspecified,
                unspecified means that there has not been any validation for this certificate before, but the host
                is absolutly the solware API server
            if (serverVerificationResult == errSecSuccess && (verificationResultType == kSecTrustResultProceed ||
                verificationResultType == kSecTrustResultUnspecified)) {

                // provide the authentication challenge with the trusted certificate
                [[authenticationChallenge sender] useCredential: [NSURLCredential credentialForTrust:
                    newTrustedCertificate] forAuthenticationChallenge: authenticationChallenge];
```

```
            } else {

                // cancel the authentication challenge, the verification could not be completed because the policy
                    could not be trusted. In other words the server connected cannot be trusted
                [[authenticationChallenge sender] cancelAuthenticationChallenge: authenticationChallenge];
            }
        } else {

            // cancel the authentication challenge because an unknown error has occured
            [[authenticationChallenge sender] cancelAuthenticationChallenge: authenticationChallenge];
        }

        // memory management release certificates and other objects
        CFRelease(certificateArray);
        CFRelease(trustedServer);
        CFRelease(newTrustedCertificate);
    }
    else {

        // cancel the current connection, an unknown authentication challenge has been received
        [[authenticationChallenge sender] cancelAuthenticationChallenge: authenticationChallenge];
    }
}

/*  Create the base 64 encoded string for the athorization header   */
- (NSString *) generateBase64AuthorizationString
{
    // when the base 64 authorization is already created return it
    if (self.base64Authorization != nil || (self.secureConnectionToken == nil && self.secureConnectionUsername ==
        nil)) {
        return self.base64Authorization;
    }

    // create the authorization string and encode it using the build in NSData base 64 encoding
    NSData *authorizationData = [[NSString stringWithFormat: @"%@:%@", self.secureConnectionUsername, self.
        secureConnectionToken] dataUsingEncoding: NSASCIIStringEncoding];

    // only use the given NSData base64 encoding methods in iOS 7, from this version on the encoding is available.
        Otherwise use own written encoding
    switch ([SBViewCompatabilityFactory cocoaVersion]) {

        // use a self written method to encode the string to base 64 representation
        case UICocoaVersionIdentifier_iOS_6_0:
        {
            // alloc memory and buffer size
            size_t bufferSize = (([authorizationData length] * 3 + 2) / 2);
            char *buffer = (char *)malloc(bufferSize);

            // get the length of the encoded base 64 char string
            int base64Length = b64_ntop([authorizationData bytes], [authorizationData length], buffer, bufferSize);

            // when failed to encode the base64Length will be -1
            if (base64Length == -1) {

                // free up the memory and return nil
                NSLog(@"Unable to create base64 representation of credentials: %@ & %@", self.secureConnectionToken,
                    self.secureConnectionUsername);
                free(buffer);

                // failed to create so return nil
                return nil;
            } else {

                // create the auth string by removing new line and whitespace characters
                self.base64Authorization = [[[NSString alloc] initWithBytesNoCopy: buffer length: bufferSize
                    encoding: NSASCIIStringEncoding freeWhenDone: YES] stringByTrimmingCharactersInSet:
                    [NSCharacterSet whitespaceAndNewlineCharacterSet]];

                return self.base64Authorization;
            }

            break;
        }

        // use the new methods available for base 64 encoding
        case UICocoaVersionIdentifier_iOS_7_0:
        case UICocoaVersionIdentifier_iOS_Future:
        {
            // encode to base 64 representation
            self.base64Authorization = [authorizationData base64EncodedStringWithOptions: 0];
            break;
        }
    }

    // return the generated string
    return self.base64Authorization;
}
```

```
/*  Validates whether the application has a valid username and token    */
- (BOOL) hasValidVerificationCredentials
{
    // when the username and token has been set return true
    if (self.secureConnectionToken != nil && self.secureConnectionUsername != nil && self.secureDeviceUUID != nil) {
        return YES;
    } else if (!isRefreshingToken){

        // the device UUID is not available, this means that a new unique device id should be created to create
            connections for
        isRefreshingToken = YES;
        self.secureDeviceUUID = [[NSUUID UUID] UUIDString];

        // create API request for the verification request and set the POST data
        SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPIAuthorizationRequestType timeout:
            60];
        [APIRequest setPOSTValue: self.secureDeviceUUID forKey: @"UUID"];

        // create and configure the operation
        SBURLOperation *authorizationOperation = [[SBURLOperation alloc] initWithAPIRequest: APIRequest options: nil
            ];
        [authorizationOperation setDelegate: self];

        // execute the operation and return that the authorization credentials are not available
        [[SBOperationManager defaultManager] addOperation: authorizationOperation];

        return NO;
    } else {

        // the refreshing operation is already being executed
        return NO;
    }
}

/*  Saves the currently fetched credentials used by the verification manager to the user defaults of the application
    */
- (void) saveCredentials
{
    // get the user defaults object
    NSUserDefaults *userDefaults = [NSUserDefaults standardUserDefaults];

    // set the new keys for the username and token
    [userDefaults setObject: self.secureConnectionUsername forKey: SBVerificationUsernameKey];
    [userDefaults setObject: self.secureConnectionToken forKey: SBVerificationTokenKey];
    [userDefaults setObject: self.secureDeviceUUID forKey: SBVerificationUUIDKey];

    // save the user defaults to disk
    if ([userDefaults synchronize]) {
    }
}

#pragma mark    -
#pragma mark    Connection delegate methods

/*  When the data has been loaded   */
- (void) operation:(SBOperation *)operation didFinishWithResult:(SBAPIResponse *) response
{
    // get the data value of the JSend response
    NSDictionary *JSONData = [response resultDictionary];

    // when the response is successfully created by the server continue parsing the auth credentials
    if ([response objectResultType] == SBAPIResultSuccessType && [JSONData objectForKey: @"token"] != nil &&
        [JSONData objectForKey: @"user"]) {

        // set the new credentials
        self.secureConnectionToken = [JSONData objectForKey: @"token"];
        self.secureConnectionUsername = [JSONData objectForKey: @"user"];

        // save the received credentials and create the authorization string
        [self saveCredentials];
        [self generateBase64AuthorizationString];

        // when the token is refresed create a new notification
        NSNotification *refreshNotification = [[NSNotification alloc] initWithName:
            SBVerificationManagerDidRefreshAuthenticationTokenNotification object: self userInfo: nil];

        // send the new notification to all listening objects
        [[NSNotificationCenter defaultCenter] postNotification: refreshNotification];
    } else {

        // create alertview that notifies the user that the initial creation of connection has failed, and make sure
            this connection can be retried
        UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
            NSLocalizedString(@"operation.authorization.request.failed.title", nil) message:
            NSLocalizedString(@"operation.authorization.request.failed.description", nil) delegate: self
            cancelButtonTitle: nil otherButtonTitles: NSLocalizedString(@"alertview.button.title.retry", nil), nil];
```

```
        // show the view
        [alertView show];
    }

    isRefreshingToken = NO;
}

/*  When the connection was not able to fetch a new username and token combination  */
- (void) operation:(SBOperation *) operation didFailWithError:(NSError *) error
{
    // create alertview that notifies the user that the initial creation of connection has failed, and make sure
        this connection can be retried
    UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
        NSLocalizedString(@"operation.authorization.request.failed.title", nil) message:
        NSLocalizedString(@"operation.authorization.request.failed.description", nil) delegate: self
        cancelButtonTitle: nil otherButtonTitles: NSLocalizedString(@"alertview.button.title.retry", nil), nil];

    // show the view
    [alertView show];
    isRefreshingToken = NO;
}

#pragma mark    -
#pragma mark    UIAlertview delegate methods

/*  When the alertview is dismissed the application should retry fetching credentials   */
- (void) alertView:(UIAlertView *) alertView willDismissWithButtonIndex:(NSInteger) buttonIndex
{
    // when not already connected retry connection
    if (!isRefreshingToken) {

        // the device UUID is not available, this means that a new unique device id should be created to create
            connections for
        isRefreshingToken = YES;
        self.secureDeviceUUID = [[NSUUID UUID] UUIDString];

        // create API request for the verification request and set the POST data
        SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPIAuthorizationRequestType timeout:
            60];
        [APIRequest setPOSTValue: self.secureDeviceUUID forKey: @"UUID"];

        // create and configure the operation
        SBURLOperation *authorizationOperation = [[SBURLOperation alloc] initWithAPIRequest: APIRequest options: nil
            ];
        [authorizationOperation setDelegate: self];

        // execute the operation and return that the authorization credentials are not available
        [[SBOperationManager defaultManager] addOperation: authorizationOperation];
    }
}

@end
```

```
//
//  Defect.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>
#import "SBManagedObjectInterface.h"

@class Element, Proceeding;

@interface Defect : NSManagedObject <SBManagedObjectInterface>

@property (nonatomic, retain) NSDate * date;
@property (nonatomic, retain) NSString * defectDescription;
@property (nonatomic, retain) NSNumber * defectId;
@property (nonatomic, retain) NSString * imageURL;
@property (nonatomic, retain) Element *element;
@property (nonatomic, retain) NSSet *proceeding;
@property (nonatomic) BOOL isSelected;

// the image fetched from the API
@property (nonatomic, strong) UIImage *fetchedImage;
@property (nonatomic) BOOL imageIsLoaded;

@end

@interface Defect (CoreDataGeneratedAccessors)

- (void)addProceedingObject:(Proceeding *)value;
- (void)removeProceedingObject:(Proceeding *)value;
- (void)addProceeding:(NSSet *)values;
- (void)removeProceeding:(NSSet *)values;

@end
```

```
//
//  Defect.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "Defect.h"
#import "Element.h"
#import "Proceeding.h"


@implementation Defect

@dynamic date;
@dynamic defectDescription;
@dynamic defectId;
@dynamic imageURL;
@dynamic element;
@dynamic proceeding;

@synthesize fetchedImage;
@synthesize imageIsLoaded;
@synthesize isSelected;

/*  Return the string representation of the managed object  */
- (NSString *) collectionViewDescription
{
    return self.defectDescription;
}

/*  Return the identifier to use for the next controller    */
- (NSNumber *) nextCategoryIdentifier
{
    return self.defectId;
}

/*  Returns the external object ID for thie managed object  */
- (NSNumber *) externalObjectID
{
    return self.defectId;
}

/*  Retrieves the image for this object */
- (UIImage *) managedObjectImage
{
    return self.fetchedImage;
}

/*  Sets the new managed object image for this object   */
- (void) setManagedObjectImage:(UIImage *)image
{
    self.fetchedImage = image;
}

/*  Whether the image is loaded */
- (BOOL) isImageLoaded
{
    return self.imageIsLoaded;
}

/*  Sets whether the image is loaded    */
- (void) setImageLoaded:(BOOL)yesNo
{
    self.imageIsLoaded = yesNo;
}

@end
```

```
//
//  Element.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>
#import "SBManagedObjectInterface.h"

@class Defect, Proceeding, Space;

@interface Element : NSManagedObject <SBManagedObjectInterface>

@property (nonatomic, retain) NSDate * date;
@property (nonatomic, retain) NSString * elementDescription;
@property (nonatomic, retain) NSNumber * elementId;
@property (nonatomic, retain) NSString * imageURL;
@property (nonatomic, retain) NSSet *defects;
@property (nonatomic, retain) Space *space;
@property (nonatomic, retain) NSSet *proceeding;
@property (nonatomic) BOOL isSelected;

// the image fetched from the API
@property (nonatomic, strong) UIImage *fetchedImage;
@property (nonatomic) BOOL imageIsLoaded;

@end

@interface Element (CoreDataGeneratedAccessors)

- (void)addDefectsObject:(Defect *)value;
- (void)removeDefectsObject:(Defect *)value;
- (void)addDefects:(NSSet *)values;
- (void)removeDefects:(NSSet *)values;

- (void)addProceedingObject:(Proceeding *)value;
- (void)removeProceedingObject:(Proceeding *)value;
- (void)addProceeding:(NSSet *)values;
- (void)removeProceeding:(NSSet *)values;

@end
```

```
//
//  Element.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "Element.h"
#import "Defect.h"
#import "Proceeding.h"
#import "Space.h"


@implementation Element

@dynamic date;
@dynamic elementDescription;
@dynamic elementId;
@dynamic imageURL;
@dynamic defects;
@dynamic space;
@dynamic proceeding;

@synthesize fetchedImage;
@synthesize imageIsLoaded;
@synthesize isSelected;

/*  Return the string representation of the managed object  */
- (NSString *) collectionViewDescription
{
    return self.elementDescription;
}

/*  Return the identifier to use for the next controller    */
- (NSNumber *) nextCategoryIdentifier
{
    return self.elementId;
}

/*  Returns the external object ID for thie managed object  */
- (NSNumber *) externalObjectID
{
    return self.elementId;
}

/*  Retrieves the image for this object */
- (UIImage *) managedObjectImage
{
    return self.fetchedImage;
}

/*  Sets the new managed object image for this object   */
- (void) setManagedObjectImage:(UIImage *)image
{
    self.fetchedImage = image;
}

/*  Whether the image is loaded */
- (BOOL) isImageLoaded
{
    return self.imageIsLoaded;
}

/*  Sets whether the image is loaded    */
- (void) setImageLoaded:(BOOL)yesNo
{
    self.imageIsLoaded = yesNo;
}

@end
```

```
//
//  Location.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>
#import "SBManagedObjectInterface.h"

@class Proceeding, Space;

@interface Location : NSManagedObject <SBManagedObjectInterface>

@property (nonatomic, retain) NSDate * date;
@property (nonatomic, retain) NSString * imageURL;
@property (nonatomic, retain) NSString * locationDescription;
@property (nonatomic, retain) NSNumber * locationId;
@property (nonatomic, retain) NSSet *spaces;
@property (nonatomic, retain) NSSet *proceeding;
@property (nonatomic) BOOL isSelected;

// the image fetched from the API
@property (nonatomic, strong) UIImage *fetchedImage;
@property (nonatomic) BOOL imageIsLoaded;

@end

@interface Location (CoreDataGeneratedAccessors)

- (void)addSpacesObject:(Space *)value;
- (void)removeSpacesObject:(Space *)value;
- (void)addSpaces:(NSSet *)values;
- (void)removeSpaces:(NSSet *)values;

- (void)addProceedingObject:(Proceeding *)value;
- (void)removeProceedingObject:(Proceeding *)value;
- (void)addProceeding:(NSSet *)values;
- (void)removeProceeding:(NSSet *)values;

@end
```

```
//
//  Location.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "Location.h"
#import "Proceeding.h"
#import "Space.h"


@implementation Location

@dynamic date;
@dynamic imageURL;
@dynamic locationDescription;
@dynamic locationId;
@dynamic spaces;
@dynamic proceeding;

@synthesize fetchedImage;
@synthesize imageIsLoaded;
@synthesize isSelected;

/*  Return the string representation of the managed object  */
- (NSString *) collectionViewDescription
{
    return self.locationDescription;
}

/*  Return the identifier to use for the next controller    */
- (NSNumber *) nextCategoryIdentifier
{
    return self.locationId;
}

/*  Returns the external object ID for thie managed object  */
- (NSNumber *) externalObjectID
{
    return self.locationId;
}

/*  Retrieves the image for this object */
- (UIImage *) managedObjectImage
{
    return self.fetchedImage;
}

/*  Sets the new managed object image for this object   */
- (void) setManagedObjectImage:(UIImage *)image
{
    self.fetchedImage = image;
}

/*  Whether the image is loaded */
- (BOOL) isImageLoaded
{
    return self.imageIsLoaded;
}

/*  Sets whether the image is loaded    */
- (void) setImageLoaded:(BOOL)yesNo
{
    self.imageIsLoaded = yesNo;
}


@end
```

```
//
//  Proceeding.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class Defect, Element, Location, Reservation, Space, SBDefectImageDescriptor;

@interface Proceeding : NSManagedObject

@property (nonatomic, retain) NSString * imageURL;
@property (nonatomic, retain) NSNumber * isUploaded;
@property (nonatomic, retain) NSString * proceedingDescription;
@property (nonatomic, retain) NSNumber * reservationProceedingIndex;
@property (nonatomic, retain) Defect *defect;
@property (nonatomic, retain) Element *element;
@property (nonatomic, retain) Location *location;
@property (nonatomic, retain) Space *space;
@property (nonatomic, retain) Reservation *reservation;


// the defect image
@property (nonatomic, strong) SBDefectImageDescriptor *imageDescriptor;

// creates an appropriate URL request for the given user, combining POST data
- (void) prepareSelectedDefectTreeForSelection:(BOOL) selected;

@end
```

```
//
//  Proceeding.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "Proceeding.h"
#import "Defect.h"
#import "Element.h"
#import "Location.h"
#import "Reservation.h"
#import "Space.h"
#import "SBDefectImageDescriptor.h"
#import "SBVerificationManager.h"
#import <AssetsLibrary/AssetsLibrary.h>

@interface Proceeding()

// appends the POST data with the given name and object
- (NSData *) generatePOSTDataForKey:(NSString *) key value:(id) value withBoundary:(NSString *) boundary;

@end

@implementation Proceeding

@dynamic reservationProceedingIndex;
@dynamic imageURL;
@dynamic isUploaded;
@dynamic proceedingDescription;
@dynamic defect;
@dynamic element;
@dynamic location;
@dynamic space;
@dynamic reservation;

@synthesize imageDescriptor;

#pragma mark    NSURLRequest generation methods

/*  Generates POST data for the given post object and key, adds a boundary  */
- (NSData *) generatePOSTDataForKey:(NSString *)key value:(id)value withBoundary:(NSString *)boundary
{
    // create the data container
    NSMutableData *partialData = [[NSMutableData alloc] init];

    // append the data for POST
    [partialData appendData: [[NSString stringWithFormat: @"--%@\r\n", boundary] dataUsingEncoding:
        NSUTF8StringEncoding]];
    [partialData appendData: [[NSString stringWithFormat: @"Content-disposition: form-data; name=\"%@\"\r\n\r\n",
        key] dataUsingEncoding: NSUTF8StringEncoding]];
    [partialData appendData: [[NSString stringWithFormat: @"%@\r\n", value] dataUsingEncoding: NSUTF8StringEncoding]
        ];

    // return the data
    return partialData;
}

/*  Sets the current defect selection tree as selected or unselected, depends on being shown or being hidden    */
- (void) prepareSelectedDefectTreeForSelection:(BOOL)selected
{
    // check each managed object for being selected
    if (self.defect != nil) {
        [self.defect setIsSelected: selected];
    }

    if (self.element != nil) {
        [self.element setIsSelected: selected];
    }

    if (self.location != nil) {
        [self.location setIsSelected: selected];
    }

    if (self.space != nil) {
        [self.space setIsSelected: selected];
    }
}
#pragma mark    -
#pragma mark    NSMAnagedObject methods

/*  When the managed object will be deleted from disk make sure that images etc are also removed    */
- (void) prepareForDeletion
{
    // call super implementation
    [super prepareForDeletion];
```

```
    // when an image is set and this image descriptor has a thumbnail, this thumbnail should be removed from disk
    if (self.imageDescriptor != nil && [self.imageDescriptor thumbnailURL] != nil) {

        // create error that will be set when image file will be removed
        NSError *error = nil;

        // get the file manager to remove the file at the given URL
        if (![[NSFileManager defaultManager] removeItemAtURL: [self.imageDescriptor thumbnailURL] error: &error]) {

            // the file could not be removed, log the error
            NSLog(@"Existing file not removed: %@", error);
        }
    }
}

@end
```

```
//
//  Reservation.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class Proceeding, SBUser;

typedef enum {
    SBReservationStatusCreating     =   1,
    SBReservationStatusUploading    =   2,
    SBReservationStatusWaiting      =   3,
    SBReservationStatusApproved     =   4,
    SBReservationStatusDenied       =   5,
    SBReservationStatusFailed       =   6,
    SBReservationStatusFinished     =   7,
    SBReservationStatusCancelled    =   8,
} SBReservationStatus;

@interface Reservation : NSManagedObject

// managed object properties which are dynamic and will be saved in the core data store
@property (nonatomic, retain) NSDate * selectedDate;
@property (nonatomic, retain) id daypart;
@property (nonatomic, retain) NSNumber * currentStatus;
@property (nonatomic, retain) NSNumber * isUploaded;
@property (nonatomic, retain) NSSet *proceedings;
@property (nonatomic, retain) NSNumber *externalReservationId;

// the user coupled to this reservation
@property (nonatomic, strong) SBUser *reservationUser;

// the status of the reservation
@property (nonatomic) NSInteger currentUploadIndex;
@property (nonatomic) BOOL isUploading;

// initializes a new reservation object
- (NSString *) localizedDate;

// get status text and text color
- (NSString *) reservationStatusDescription;
- (UIColor *) reservationStatusTextColor;

// get other descriptions for cell labels
- (NSString *) reservationDefectDescription;
- (NSString *) reservationDateDescription;
- (NSString *) reservationDayPartDescription;

@end

// accessor methods for core data
@interface Reservation (CoreDataGeneratedAccessors)

- (void)addProceedingsObject:(Proceeding *)value;
- (void)removeProceedingsObject:(Proceeding *)value;
- (void)addProceedings:(NSSet *)values;
- (void)removeProceedings:(NSSet *)values;


@end
```

```
//
//  Reservation.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "Reservation.h"
#import "Proceeding.h"
#import "SBDayPartDescriptor.h"
#import "Defect.h"
#import "SBUser.h"
#import "SBVerificationManager.h"


@implementation Reservation

@dynamic selectedDate;
@dynamic daypart;
@dynamic currentStatus;
@dynamic isUploaded;
@dynamic proceedings;
@dynamic externalReservationId;

@synthesize reservationUser;
@synthesize currentUploadIndex;
@synthesize isUploading;

/*  Override the creation of the managed object to do initialization for this reservation object    */
- (id) initWithEntity:(NSEntityDescription *)entity insertIntoManagedObjectContext:(NSManagedObjectContext *)context
{
    // call super to create the actual entity for reservation
    self = [super initWithEntity: entity insertIntoManagedObjectContext: context];

    if (self) {

        // create the user and fetch the optional data from user defaults
        self.reservationUser = [SBUser sharedUser];

        // set the default values for unknown descriptions etc.
        if (self.currentStatus == nil) {
            self.currentStatus = [NSNumber numberWithInteger: SBReservationStatusCreating];
        }
    }

    return self;
}

/*  This method returns the correct timezone formatted date string, an NSDate object is always in GMT time, which
    can be 2 hours off the date that is actually selected */
- (NSString *) localizedDate
{
    if (self.selectedDate != nil) {
        // create the date formatter and configure it for the correct format
        NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
        [dateFormatter setDateFormat: @"Y-M-d"];

        // return the timezone localized string
        return [dateFormatter stringFromDate: self.selectedDate];
    }
    else {
        // return localized stirng for unknown date
        return NSLocalizedString(@"reservation.object.unknown.date.description", nil);
    }
}

/*  Retrieve the formatted date description for the overview cell of this reservation   */
- (NSString *) reservationDateDescription
{
    if (self.selectedDate != nil) {
        // create date formatter for the following format (monday, 21 September 2013)
        NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
        [dateFormatter setLocale: [NSLocale localeWithLocaleIdentifier: @"nl_NL"]];
        [dateFormatter setDateFormat: @"EEEE, d MMMM y"];

        // return the formatted date
        return [dateFormatter stringFromDate: self.selectedDate];
    }
    else {
        return NSLocalizedString(@"reservation.object.unknown.date.description", nil);
    }
}

/*  Get the description string for the reservation ay part  */
- (NSString *) reservationDayPartDescription
{
```

```
    if (self.daypart != nil) {
        // cast the day part object as a day part descriptor
        SBDayPartDescriptor *dayPartDescriptorCast = (SBDayPartDescriptor *)self.daypart;

        // return the time description of the day part object
        return [dayPartDescriptorCast timeDescription];
    }
    else {
        return NSLocalizedString(@"reservation.object.unknown.daypart.description", nil);
    }
}

/*  Returns a string object identifyig the status of this reservation   */
- (NSString *) reservationStatusDescription
{
    // create default status description
    NSString *statusDescription = nil;

    // switch to find the corresponding text description for the current status of the reservation
    switch ([self.currentStatus integerValue]) {
        case SBReservationStatusApproved:
            statusDescription = NSLocalizedString(@"reservation.status.approved.title", nil);
            break;

        case SBReservationStatusCreating:
            statusDescription = NSLocalizedString(@"reservation.status.creating.title", nil);
            break;

        case SBReservationStatusDenied:
            statusDescription = NSLocalizedString(@"reservation.status.denied.title", nil);
            break;

        case SBReservationStatusUploading:
            statusDescription = NSLocalizedString(@"reservation.status.uploading.title", nil);
            break;

        case SBReservationStatusWaiting:
            statusDescription = NSLocalizedString(@"reservation.status.waiting.title", nil);
            break;

        case SBReservationStatusFailed:
            statusDescription = NSLocalizedString(@"reservation.status.failed.title", nil);
            break;

        case SBReservationStatusFinished:
            statusDescription = NSLocalizedString(@"reservation.status.finished.title", nil);
            break;

        case SBReservationStatusCancelled:
            statusDescription = NSLocalizedString(@"reservation.status.cancelled.title", nil);
            break;
    }

    return statusDescription;
}

/*  Retreives the defect description of the first proceeding in this reservation    */
- (NSString *) reservationDefectDescription
{
    // get the total amount of proceedings for this reservation
    NSInteger totalProceedings = [self.proceedings count];
    NSString *defectDescription = nil;

    // get the defect object by managed object id, this way we always have the most up to date defect object
    Defect *currentDefect = [[self.proceedings anyObject] defect];

    // when the current defetc is avialable show the defect description
    if (currentDefect != nil) {
        if (totalProceedings > 1) {

            // create the description for multiple reservation proceedings
            defectDescription = [NSString stringWithFormat: @"%@ (%d)", [currentDefect defectDescription],
                totalProceedings];
        }
        else {
            defectDescription = [NSString stringWithFormat: @"%@", [currentDefect defectDescription]];
        }
    }
    else {
        // set the single discription for the reservation
        defectDescription = NSLocalizedString(@"reservation.object.unknown.proceeding.description", nil);
    }

    return defectDescription;
}

/*  Retrieve the text color of the status text label */
```

```
- (UIColor *) reservationStatusTextColor
{
    // create default color
    UIColor *statusColor = nil;

    // switch to find the corresponding text color for the description status string
    switch ([self.currentStatus integerValue]) {
        case SBReservationStatusApproved:
            statusColor = [UIColor greenColor];
            break;

        case SBReservationStatusCreating:
            statusColor = [UIColor yellowColor];
            break;

        case SBReservationStatusDenied:
            statusColor = [UIColor redColor];
            break;

        case SBReservationStatusUploading:
            statusColor = [UIColor orangeColor];
            break;

        case SBReservationStatusWaiting:
            statusColor = [UIColor yellowColor];
            break;

        case SBReservationStatusFailed:
            statusColor = [UIColor redColor];
            break;

        case SBReservationStatusCancelled:
            statusColor = [UIColor redColor];
            break;

        case SBReservationStatusFinished:
            statusColor = [UIColor greenColor];
            break;
    }

    return statusColor;
}

#pragma mark    -

@end
```

```
//
//  Space.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>
#import "SBManagedObjectInterface.h"

@class Element, Location, Proceeding;

@interface Space : NSManagedObject <SBManagedObjectInterface>

@property (nonatomic, retain) NSDate * date;
@property (nonatomic, retain) NSString * imageURL;
@property (nonatomic, retain) NSString * spaceDescription;
@property (nonatomic, retain) NSNumber * spaceId;
@property (nonatomic, retain) NSSet *elements;
@property (nonatomic, retain) Location *location;
@property (nonatomic, retain) NSSet *proceeding;
@property (nonatomic) BOOL isSelected;

// the image fetched from the API
@property (nonatomic, strong) UIImage *fetchedImage;
@property (nonatomic) BOOL imageIsLoaded;

@end

@interface Space (CoreDataGeneratedAccessors)

- (void)addElementsObject:(Element *)value;
- (void)removeElementsObject:(Element *)value;
- (void)addElements:(NSSet *)values;
- (void)removeElements:(NSSet *)values;

- (void)addProceedingObject:(Proceeding *)value;
- (void)removeProceedingObject:(Proceeding *)value;
- (void)addProceeding:(NSSet *)values;
- (void)removeProceeding:(NSSet *)values;

@end
```

```
//
//  Space.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 30-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "Space.h"
#import "Element.h"
#import "Location.h"
#import "Proceeding.h"


@implementation Space

@dynamic date;
@dynamic imageURL;
@dynamic spaceDescription;
@dynamic spaceId;
@dynamic elements;
@dynamic location;
@dynamic proceeding;

@synthesize fetchedImage;
@synthesize imageIsLoaded;
@synthesize isSelected;

/*  Return the string representation of the managed object  */
- (NSString *) collectionViewDescription
{
    return self.spaceDescription;
}

/*  Return the identifier to use for the next controller    */
- (NSNumber *) nextCategoryIdentifier
{
    return self.spaceId;
}

/*  Returns the external object ID for thie managed object  */
- (NSNumber *) externalObjectID
{
    return self.spaceId;
}

/*  Retrieves the image for this object */
- (UIImage *) managedObjectImage
{
    return self.fetchedImage;
}

/*  Sets the new managed object image for this object   */
- (void) setManagedObjectImage:(UIImage *)image
{
    self.fetchedImage = image;
}

/*  Whether the image is loaded */
- (BOOL) isImageLoaded
{
    return self.imageIsLoaded;
}

/*  Sets whether the image is loaded    */
- (void) setImageLoaded:(BOOL)yesNo
{
    self.imageIsLoaded = yesNo;
}


@end
```

```
//
//  SBCollectionCellCompatabilityView_iOS_6_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBCollectionCellCompatabilityView.h"

@interface SBCollectionCellCompatabilityView_iOS_6_0 : SBCollectionCellCompatabilityView

@end
```

```
//
//  SBCollectionCellCompatabilityView_iOS_6_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBCollectionCellCompatabilityView_iOS_6_0.h"
#import "SBElementCollectionViewCell.h"
#import <QuartzCore/QuartzCore.h>

@implementation SBCollectionCellCompatabilityView_iOS_6_0

/*  Initializes the given collection view cell and prepares the text color etc for a specific iOS   */
- (void) initializeCollectionViewCell:(SBElementCollectionViewCell *)collectionCell inFrame:(CGRect)frame
{
    // create the textlabe with the width and height of the view that will be created
    UIFont *newFont = [UIFont fontWithName: @"AppleGothic" size: 12];

    // create the element label with the font settings, center alignment and red color for now
    collectionCell.elementLabel = [[UILabel alloc] initWithFrame: CGRectMake(0, 0, frame.size.width, frame.size.
        height)];
    [collectionCell.elementLabel setTextAlignment: NSTextAlignmentCenter];
    [collectionCell.elementLabel setTextColor: [UIColor blackColor]];
    [collectionCell.elementLabel setBackgroundColor: [UIColor clearColor]];
    [collectionCell.elementLabel setFont: newFont];
    [collectionCell.elementLabel setLineBreakMode: NSLineBreakByWordWrapping];
    [collectionCell.elementLabel setNumberOfLines: 4];

    // add the element label textview to the view
    [collectionCell addSubview: collectionCell.elementLabel];
}

/*  Implements the drawRect method that is passed to the element view cell, redraws the frame   */
- (void) redrawCollectionCell:(SBElementCollectionViewCell *) cell inRect:(CGRect) frame;
{
    // get label size
    CGSize labelSize = [[[cell elementLabel] text] sizeWithFont: [cell.elementLabel font] constrainedToSize:
        CGSizeMake(frame.size.width, frame.size.height) lineBreakMode: NSLineBreakByCharWrapping];

    // calculate the middel coordinates for this view
    CGFloat x = (frame.size.width - labelSize.width) / 2;
    CGFloat y = (frame.size.height - labelSize.height) / 2;

    // set the frame and number of lines for this label
    [cell.elementLabel setFrame: CGRectMake(x, y, labelSize.width, labelSize.height)];

    // give the layer a white color around the view with a width of 3 points
    cell.layer.borderColor = [UIColor grayColor].CGColor;
    cell.layer.borderWidth = 1.0;

    // create a shadow under the layer, of course is black with 3 points width and half opacity
    cell.layer.shadowColor = [UIColor blackColor].CGColor;
    cell.layer.shadowRadius = 3.0;
    cell.layer.shadowOpacity = 0.5;

    // offset of the shadow is 2 points in the height, also create a shadow path which makes rendering faster
    cell.layer.shadowOffset = CGSizeMake(0.0, 2.0);
    CGPathRef shadowPath = CGPathCreateWithRect(cell.bounds, NULL);
    cell.layer.shadowPath = shadowPath;

    // release the shadow path
    CGPathRelease(shadowPath);
}

@end
```

```
//
//  SBCollectionCellCompatabilityView_iOS_7_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBCollectionCellCompatabilityView.h"

@interface SBCollectionCellCompatabilityView_iOS_7_0 : SBCollectionCellCompatabilityView


@end
```

```
//
//  SBCollectionCellCompatabilityView_iOS_7_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBCollectionCellCompatabilityView_iOS_7_0.h"
#import "SBElementCollectionViewCell.h"
#import <QuartzCore/QuartzCore.h>

@implementation SBCollectionCellCompatabilityView_iOS_7_0

/*  Initialize the font nd background color of the collection view cell */
- (void) initializeCollectionViewCell:(SBElementCollectionViewCell *) collectionCell inFrame:(CGRect) frame
{
    // create the textlabe with the width and height of the view that will be created
    //UIFont *newFont = [UIFont fontWithName: @"AppleGothic" size: 12];
    UIFont *newFont = [UIFont systemFontOfSize: [UIFont systemFontSize]];

    // create the element label with the font settings, center alignment and red color for now
    collectionCell.elementLabel = [[UILabel alloc] initWithFrame: CGRectMake(0, 0, frame.size.width, frame.size.
        height)];
    [collectionCell.elementLabel setTextAlignment: NSTextAlignmentCenter];
    [collectionCell.elementLabel setTextColor: [UIColor blackColor]];
    [collectionCell.elementLabel setFont: newFont];
    [collectionCell.elementLabel setLineBreakMode: NSLineBreakByWordWrapping];

    // add the element label textview to the view
    [collectionCell addSubview: collectionCell.elementLabel];
}

/*  Drawing code for iOS 7 and the redraw rect method   */
- (void) redrawCollectionCell:(SBElementCollectionViewCell *) cell inRect:(CGRect) frame;
{
    // create and configure the string drawing context for calculating the text size
    /*NSStringDrawingContext *drawContext = [[NSStringDrawingContext alloc] init];

    // calculate the frame of the textlabel
    CGRect labelSize = [[[cell elementLabel] text] boundingRectWithSize: CGSizeMake(frame.size.width - 5,
        frame.size.height - 5) options: NSStringDrawingUsesLineFragmentOrigin attributes: [NSDictionary
        dictionaryWithObject: [cell.elementLabel font] forKey: NSFontAttributeName] context: drawContext];

    // calculate the middel coordinates for this view
    CGFloat x = (frame.size.width - labelSize.size.width) / 2;
    CGFloat y = (frame.size.height - labelSize.size.height) / 2;

    // set the frame and number of lines for this label
    [cell.elementLabel setFrame: CGRectMake(x, y, labelSize.size.width, labelSize.size.height)];
    [cell.elementLabel setNumberOfLines: 4];

    // give the layer a white color around the view with a width of 3 points
    cell.layer.borderColor = [UIColor grayColor].CGColor;
    cell.layer.borderWidth = 1.0;

    // create a shadow under the layer, of course is black with 3 points width and half opacity
    cell.layer.shadowColor = [UIColor blackColor].CGColor;
    cell.layer.shadowRadius = 3.0;
    cell.layer.shadowOpacity = 0.5;

    // offset of the shadow is 2 points in the height, also create a shadow path which makes rendering faster
    cell.layer.shadowOffset = CGSizeMake(0.0, 2.0);
    CGPathRef shadowPath = CGPathCreateWithRoundedRect(cell.bounds, 0.0, 2.0,  NULL);
    cell.layer.shadowPath = shadowPath;

    // release the shadow path
    CGPathRelease(shadowPath);*/
}

+ (CGSize) calculateCellSizeForString:(NSString *)string
{
    // create and configure the string drawing context for calculating the text size
    NSStringDrawingContext *drawContext = [[NSStringDrawingContext alloc] init];

    // calculate the frame of the textlabel
    CGRect labelSize = [string boundingRectWithSize: CGSizeMake(75, 115) options:
        NSStringDrawingUsesLineFragmentOrigin attributes: [NSDictionary dictionaryWithObject: [UIFont
        systemFontOfSize: [UIFont systemFontSize]] forKey: NSFontAttributeName] context: drawContext];

    return CGSizeMake(labelSize.size.width, labelSize.size.height);
}

@end
```

```
//
//  SBCollectionCellCompatabilityView.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBCompatabilityInterface.h"
@class SBElementCollectionViewCell;

@interface SBCollectionCellCompatabilityView : NSObject <SBCompatabilityInterface>

// draw the collection cell
- (void) redrawCollectionCell:(SBElementCollectionViewCell *) cell inRect:(CGRect) frame;
+ (CGSize) calculateCellSizeForString:(NSString *) string;

@end
```

```
//
//  SBCollectionCellCompatabilityView.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBCollectionCellCompatabilityView.h"
#import "SBCompatabilityInterface.h"
#import "SBElementCollectionViewCell.h"

@implementation SBCollectionCellCompatabilityView

// declare abstract methods, not implementd here will be implemented by individual iOS classes
- (void) initializeInterfaceOfViewController:(UIViewController *)viewController{}
- (void) redrawCollectionCell:(SBElementCollectionViewCell *) cell inRect:(CGRect) frame{}

+ (CGSize) calculateCellSizeForString:(NSString *) string{return CGSizeZero;}

@end
```

```
//
//  SBErrorCompatabilityView_iOS_6_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBErrorCompatabilityView.h"

@interface SBErrorCompatabilityView_iOS_6_0 : SBErrorCompatabilityView

@end
```

```
//
//  SBErrorCompatabilityView_iOS_6_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBErrorCompatabilityView_iOS_6_0.h"
#import "SBErrorView.h"

@implementation SBErrorCompatabilityView_iOS_6_0

- (void) drawRect:(CGRect)frame forErrorViewController:(SBErrorView *)errorView
{
    // the options for drawing the string
    UIFont *stringFont = [UIFont systemFontOfSize: [UIFont labelFontSize]];

    // get the rect used for this drawing context and size
    CGSize messageRect = [errorView.errorMessage sizeWithFont: stringFont constrainedToSize: CGSizeMake(frame.size.
        width, frame.size.height) lineBreakMode: NSLineBreakByWordWrapping];
    // get the additional view for the image type and add it to the view when not nil
    UIView *additionalView = [errorView returnImageViewForMessageRect: CGRectMake(0, 0, messageRect.width,
        messageRect.height)];

    if (additionalView != nil){
        [errorView addSubview: additionalView];
    }

    // now we now the height and width of the string, caculate the middle of the view
    CGFloat x = (frame.size.width - messageRect.width) / 2;
    CGFloat y = (frame.size.height - messageRect.height) / 2;

    // create the label with the calculated size
    UILabel *textLabel = [[UILabel alloc] initWithFrame: CGRectMake(x, y, messageRect.width, messageRect.height)];
    [textLabel setText: errorView.errorMessage];
    [textLabel setFont: stringFont];
    [textLabel setLineBreakMode: NSLineBreakByWordWrapping];
    [textLabel setNumberOfLines: (messageRect.height / [stringFont lineHeight])];
    [textLabel setTextAlignment: NSTextAlignmentCenter];
    [textLabel setTextColor: [UIColor grayColor]];

    // add to the view
    [errorView addSubview: textLabel];
}

@end
```

```
//
//  SBErrorCompatabilityView_iOS_7_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBErrorCompatabilityView.h"

@interface SBErrorCompatabilityView_iOS_7_0 : SBErrorCompatabilityView

@end
```

```
//
//  SBErrorCompatabilityView_iOS_7_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBErrorCompatabilityView_iOS_7_0.h"
#import "SBErrorView.h"

@implementation SBErrorCompatabilityView_iOS_7_0

- (void) drawRect:(CGRect)frame forErrorViewController:(SBErrorView *)errorView
{
    // calculate the width and height of the string that will be drawn
    NSStringDrawingContext *drawingContext = [[NSStringDrawingContext alloc] init];
    [drawingContext setMinimumScaleFactor: 0];

    // the options for drawing the string
    UIFont *stringFont = [UIFont systemFontOfSize: [UIFont labelFontSize]];
    NSDictionary *drawingOptions = [NSDictionary dictionaryWithObjectsAndKeys: stringFont, NSFontAttributeName, nil]
        ;

    // get the rect used for this drawing context and size
    CGRect messageRect = [errorView.errorMessage boundingRectWithSize: CGSizeMake(300, 100) options:
        NSStringDrawingUsesLineFragmentOrigin attributes: drawingOptions context: drawingContext];

    // get the additional view for the image type and add it to the view when not nil
    UIView *additionalView = [errorView returnImageViewForMessageRect: messageRect];

    // when available set the additonal view for the error view
    if (additionalView != nil) {
        [errorView addSubview: additionalView];
    }

    // now we now the height and width of the string, caculate the middle of the view
    CGFloat x = (frame.size.width - messageRect.size.width) / 2;
    CGFloat y = (frame.size.height - messageRect.size.height) / 2;

    // create the label with the calculated size
    UILabel *textLabel = [[UILabel alloc] initWithFrame: CGRectMake(x, y, messageRect.size.width, messageRect.size.
        height)];
    [textLabel setText: errorView.errorMessage];
    [textLabel setFont: stringFont];
    [textLabel setLineBreakMode: NSLineBreakByWordWrapping];
    [textLabel setNumberOfLines: (messageRect.size.height / [stringFont lineHeight])];
    [textLabel setTextAlignment: NSTextAlignmentCenter];
    [textLabel setTextColor: [UIColor grayColor]];

    // add to the view
    [errorView addSubview: textLabel];
}

@end
```

```
//
//  SBErrorCompatabilityView.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@class SBErrorView;

@interface SBErrorCompatabilityView : NSObject

- (void) drawRect:(CGRect) frame forErrorViewController:(SBErrorView *) errorView;

@end
```

```
//
//  SBErrorCompatabilityView.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBErrorCompatabilityView.h"

@implementation SBErrorCompatabilityView

// draws the error view controller contents in the specified frame
- (void) drawRect:(CGRect)frame forErrorViewController:(SBErrorView *)errorView{}

@end
```

```
//
//  SBImageCompatabilityView_iOS_6_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBImageCompatabilityView.h"

@interface SBImageCompatabilityView_iOS_6_0 : SBImageCompatabilityView

@end
```

```
//
//  SBImageCompatabilityView_iOS_6_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBImageCompatabilityView_iOS_6_0.h"
#import "SBPhoneImageViewController.h"
#import <QuartzCore/QuartzCore.h>

@implementation SBImageCompatabilityView_iOS_6_0

- (void) drawImageSelectionButtonForController:(SBPhoneImageViewController *)viewController
{
    // set the button title localized and create the context for the string drawing
    NSString *buttonTitle = NSLocalizedString(@"image.controller.button.select.photo.title", nil);

    // get the size of the button title string
    CGSize calculatedSize = [buttonTitle sizeWithFont: [UIFont fontWithName: @"AppleGothic" size: 15]
        constrainedToSize: CGSizeMake(viewController.view.frame.size.width, viewController.view.frame.size.height)
        lineBreakMode: NSLineBreakByWordWrapping];

    // declare X & Y
    CGFloat x = 0, y = 0;
    x = (viewController.view.bounds.size.width - calculatedSize.width) / 2;

    // when an image is set place the button below the image
    if ([viewController.imageDescriptor imageThumbnail] != nil)
    {
        // get the height of the image
        NSInteger imageHeight = [viewController.imageDescriptor imageThumbnail].size.height;
        y = viewController.view.frame.size.height - ((viewController.view.frame.size.height - imageHeight) / 4);
    }
    else
    {
        // calculate the x & y coordinates of the button to be in the middle of the screen
        y = (viewController.view.bounds.size.height - calculatedSize.height) / 2;
    }

    // create the button and set the text
    viewController.imageSelectionButton = [UIButton buttonWithType: UIButtonTypeCustom];

    // configure the button
    [viewController.imageSelectionButton setTitle: buttonTitle forState: UIControlStateNormal];
    [[viewController.imageSelectionButton titleLabel] setFont: [UIFont fontWithName: @"AppleGothic" size:15]];
    [viewController.imageSelectionButton setTitleColor: [UIColor blueColor] forState: UIControlStateNormal];

    // set the button frame
    viewController.imageSelectionButton.frame = CGRectMake(x, y, calculatedSize.width, calculatedSize.height);
    [viewController.imageSelectionButton addTarget: viewController action: @selector(photoSelectionButtonTapped:)
        forControlEvents: UIControlEventTouchDown];

    // add the button to the view
    [viewController.view addSubview: viewController.imageSelectionButton];
}

/*  Draw the image on the image controller screen for iOS 6 */
- (void) drawImageInControllerView:(SBPhoneImageViewController *)viewController
{
    // create the image view with the created thumbnail
    UIImageView *imageview = [[UIImageView alloc] initWithImage: [viewController.imageDescriptor imageThumbnail]];

    // calculate the middle of the view
    CGFloat x = (viewController.view.frame.size.width - [viewController.imageDescriptor imageThumbnail].size.width)
        / 2;
    CGFloat y = (viewController.view.frame.size.height - [viewController.imageDescriptor imageThumbnail].size.height
        ) / 2;

    // set the image frame and add to the view
    [imageview setFrame: CGRectMake(x, y, [viewController.imageDescriptor imageThumbnail].size.width,
        [viewController.imageDescriptor imageThumbnail].size.height)];

    // give the layer a white color around the view with a width of 3 points
    imageview.layer.borderColor = [UIColor orangeColor].CGColor;
    imageview.layer.borderWidth = 3.0;

    // create a shadow under the layer, of course is black with 3 points width and half opacity
    imageview.layer.shadowColor = [UIColor blackColor].CGColor;
    imageview.layer.shadowRadius = 3.0;
    imageview.layer.shadowOpacity = 0.5;

    // offset of the shadow is 2 points in the height, also create a shadow path which makes rendering faster
    imageview.layer.shadowOffset = CGSizeMake(0.0, 2.0);
    CGPathRef shadowPath = CGPathCreateWithRect(imageview.bounds, NULL);
    imageview.layer.shadowPath = shadowPath;
```

```
    // release the shadow path
    CGPathRelease(shadowPath);
    [viewController.view addSubview: imageview];
}

@end
```

```
//
//  SBImageCompatabilityView_iOS_7_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBImageCompatabilityView.h"

@interface SBImageCompatabilityView_iOS_7_0 : SBImageCompatabilityView

@end
```

```
//
//  SBImageCompatabilityView_iOS_7_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBImageCompatabilityView_iOS_7_0.h"
#import "SBPhoneImageViewController.h"
#import <QuartzCore/QuartzCore.h>

@implementation SBImageCompatabilityView_iOS_7_0

/*  Draws the button on the screen  */
- (void) drawImageSelectionButtonForController:(SBPhoneImageViewController *)viewController
{
    // set the button title localized and create the context for the string drawing
    NSString *buttonTitle = NSLocalizedString(@"image.controller.button.select.photo.title", nil);
    NSStringDrawingContext *drawingContext = [[NSStringDrawingContext alloc] init];

    // configure the drawing context
    [drawingContext setMinimumScaleFactor: 1];
    CGRect calculatedRect = [buttonTitle boundingRectWithSize: CGSizeMake(viewController.view.bounds.size.width, 44)
        options: NSStringDrawingUsesFontLeading attributes: [NSDictionary dictionaryWithObject: [UIFont
        fontWithName: @"AppleGothic" size:15] forKey: NSFontAttributeName] context: drawingContext];

    // declare X & Y
    CGFloat x = 0, y = 0;
    x = (viewController.view.bounds.size.width - calculatedRect.size.width) / 2;

    // when an image is set place the button below the image
    if ([viewController.imageDescriptor imageThumbnail] != nil)
    {
        // get the height of the image
        NSInteger imageHeight = [viewController.imageDescriptor imageThumbnail].size.height;
        y = viewController.view.frame.size.height - ((viewController.view.frame.size.height - imageHeight) / 4);
    }
    else
    {
        // calculate the x & y coordinates of the button to be in the middle of the screen
        y = (viewController.view.bounds.size.height - calculatedRect.size.height) / 2;
    }

    // create the button and set the text
    viewController.imageSelectionButton = [UIButton buttonWithType: UIButtonTypeRoundedRect];

    // configure the button
    [viewController.imageSelectionButton setTitle: buttonTitle forState: UIControlStateNormal];
    [[viewController.imageSelectionButton titleLabel] setFont: [UIFont fontWithName: @"AppleGothic" size:15]];
    viewController.imageSelectionButton.frame = CGRectMake(x, y, calculatedRect.size.width, calculatedRect.size.
        height);
    [viewController.imageSelectionButton addTarget: viewController action: @selector(photoSelectionButtonTapped:)
        forControlEvents: UIControlEventTouchDown];

    // add the button to the view
    [viewController.view addSubview: viewController.imageSelectionButton];
}

- (void) drawImageInControllerView:(SBPhoneImageViewController *)viewController
{
    // create the image view with the created thumbnail
    UIImageView *imageview = [[UIImageView alloc] initWithImage: [viewController.imageDescriptor imageThumbnail]];
    [imageview setContentMode: UIViewContentModeScaleAspectFit];

    // get the image size
    CGSize imageSize = [[viewController.imageDescriptor imageThumbnail] size];

    // cap the image sizes
    if (imageSize.width > viewController.view.frame.size.width) {

        CGFloat resize = (imageSize.width - viewController.view.frame.size.width) / imageSize.width;
        imageSize.width = viewController.view.frame.size.width;
        imageSize.height = (imageSize.height * (1 - resize));
    } else if (imageSize.height > (viewController.view.frame.size.height - 50)) {

        CGFloat resize = (imageSize.height - (viewController.view.frame.size.height - 50)) / imageSize.height;

        imageSize.height = (viewController.view.frame.size.height - 50);
        imageSize.width = (imageSize.width * (1 - resize));
    }

    // calculate the middle of the view
    CGFloat x = (viewController.view.frame.size.width - imageSize.width) / 2;
    CGFloat y = (viewController.view.frame.size.height - imageSize.height) / 2;

    // set the image frame and add to the view
```

```
    [imageview setFrame: CGRectMake(x, y, imageSize.width, imageSize.height)];

    // give the layer a white color around the view with a width of 3 points
    //imageview.layer.borderColor = [UIColor orangeColor].CGColor;
    //imageview.layer.borderWidth = 3.0;

    // create a shadow under the layer, of course is black with 3 points width and half opacity
    imageview.layer.shadowColor = [UIColor blackColor].CGColor;
    imageview.layer.shadowRadius = 3.0;
    imageview.layer.shadowOpacity = 0.5;

    // offset of the shadow is 2 points in the height, also create a shadow path which makes rendering faster
    imageview.layer.shadowOffset = CGSizeMake(0.0, 2.0);
    CGPathRef shadowPath = CGPathCreateWithRoundedRect(imageview.bounds, 0.0, 2.0,  NULL);
    imageview.layer.shadowPath = shadowPath;

    // release the shadow path
    CGPathRelease(shadowPath);
    [viewController.view addSubview: imageview];
}

@end
```

```
//
//  SBImagecompatabilityView.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBCompatabilityInterface.h"

@class SBPhoneImageViewController;

@interface SBImageCompatabilityView : NSObject <SBCompatabilityInterface>

// initializer for image controller
- (void) initializeInterfaceOfViewController:(SBPhoneImageViewController *)viewController;
- (void) drawImageSelectionButtonForController:(SBPhoneImageViewController *) viewController;
- (void) drawImageInControllerView:(SBPhoneImageViewController *) viewController;

@end
```

```
//
//  SBImagecompatabilityView.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBImageCompatabilityView.h"

@implementation SBImageCompatabilityView

- (void) initializeInterfaceOfViewController:(SBPhoneImageViewController *)viewController{}
- (void) drawImageSelectionButtonForController:(SBPhoneImageViewController *)viewController{}
- (void) drawImageInControllerView:(SBPhoneImageViewController *)viewController{}

@end
```

```
//
//  SBKeyboardToolbarCompatabilityView_iOS_6_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 13-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBKeyboardToolbarCompatabilityView.h"

@interface SBKeyboardToolbarCompatabilityView_iOS_6_0 : SBKeyboardToolbarCompatabilityView

@end
```

```
//
//  SBKeyboardToolbarCompatabilityView_iOS_6_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 13-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBKeyboardToolbarCompatabilityView_iOS_6_0.h"
#import "SBKeyboardCustomToolbar.h"

@implementation SBKeyboardToolbarCompatabilityView_iOS_6_0

- (void) initializeInterfaceOfViewController:(SBKeyboardCustomToolbar *)viewController
{
    // for iOS 6 set the text color to white and add shadow
    [[viewController characterCountLabel] setTextColor: [UIColor whiteColor]];
    [[viewController characterCountLabel] setShadowColor: [UIColor darkGrayColor]];
    [[viewController characterCountLabel] setShadowOffset: CGSizeMake(0, -1)];
}

@end
```

```
//
//  SBKeyboardToolbarCompatabilityView_iOS_7_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 13-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBKeyboardToolbarCompatabilityView.h"

@interface SBKeyboardToolbarCompatabilityView_iOS_7_0 : SBKeyboardToolbarCompatabilityView

@end
```

```
//
//  SBKeyboardToolbarCompatabilityView_iOS_7_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 13-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBKeyboardToolbarCompatabilityView_iOS_7_0.h"
#import "SBKeyboardCustomToolbar.h"

@implementation SBKeyboardToolbarCompatabilityView_iOS_7_0

/*  Set the text color of the label in iOS 7    */
- (void) initializeInterfaceOfViewController:(SBKeyboardCustomToolbar *)viewController
{
    // set the text color to black because of the white / gray background
    [viewController.characterCountLabel setTextColor: [UIColor blackColor]];
}

@end
```

```
//
//  SBKeyboardToolbarCompatabilityView.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 13-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBCompatabilityInterface.h"

@class SBKeyboardCustomToolbar;

@interface SBKeyboardToolbarCompatabilityView : NSObject <SBCompatabilityInterface>

- (void) initializeInterfaceOfViewController:(SBKeyboardCustomToolbar *)viewController;

@end
```

```m
//
//  SBKeyboardToolbarCompatabilityView.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 13-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBKeyboardToolbarCompatabilityView.h"

@implementation SBKeyboardToolbarCompatabilityView

// create empty definition
- (void) initializeInterfaceOfViewController:(SBKeyboardToolbarCompatabilityView *)viewController{}

@end
```

```
//
//  SBLocationCompatabilityView_iOS6_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBLocationCompatabilityView.h"

@interface SBLocationCompatabilityView_iOS_6_0 : SBLocationCompatabilityView

@end
```

```
//
//  SBLocationCompatabilityView_iOS6_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBLocationCompatabilityView_iOS_6_0.h"
#import "SBPhoneLocationViewController.h"

@implementation SBLocationCompatabilityView_iOS_6_0

/*  Initialize the view controller for iOS 6.0 or 6.1   */
- (void) initializeInterfaceOfViewController:(SBPhoneLocationViewController *)viewController
{

}

@end
```

```
//
//  SBLocationCompatabilityView_iOS_7_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBLocationCompatabilityView.h"

@interface SBLocationCompatabilityView_iOS_7_0 : SBLocationCompatabilityView

@end
```

```
//
//  SBLocationCompatabilityView_iOS_7_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBLocationCompatabilityView_iOS_7_0.h"
#import "SBPhoneLocationViewController.h"

@implementation SBLocationCompatabilityView_iOS_7_0

/*  Initialize all version specific user interface elements in this function, usally should be called from the view
    did load method */
- (void) initializeInterfaceOfViewController:(SBPhoneLocationViewController *)viewController
{

}

@end
```

```
//
//  SBLocationCompatabilityView.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBCompatabilityInterface.h"

@class SBPhoneLocationViewController;

@interface SBLocationCompatabilityView : NSObject <SBCompatabilityInterface>

// class specific methods
- (void) initializeInterfaceOfViewController:(SBPhoneLocationViewController *)viewController;

@end
```

```
//
//  SBLocationCompatabilityView.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBLocationCompatabilityView.h"

@implementation SBLocationCompatabilityView

// initialize methods with empty implementation, abstract class
- (void) initializeInterfaceOfViewController:(UIViewController *)viewController{}

@end
```

```
//
//  SBLoginCompatabilityView_iOS_6_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBLoginCompatabilityView.h"

@interface SBLoginCompatabilityView_iOS_6_0 : SBLoginCompatabilityView

@end
```

```
//
//  SBLoginCompatabilityView_iOS_6_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBLoginCompatabilityView_iOS_6_0.h"
#import "SBPhoneLoginViewController.h"

@implementation SBLoginCompatabilityView_iOS_6_0

/*  Initialize the controller by setting the bar button and the correct frames for the textfields per version   */
- (void) initializeInterfaceOfViewController:(SBPhoneLoginViewController *)viewController
{
    // create a plus button and add it to the navigation bar
    UIBarButtonItem *addMaintenanceButton = [[UIBarButtonItem alloc] initWithTitle:
        NSLocalizedString(@"login.controller.navigation.done.title", nil) style: UIBarButtonItemStyleDone target:
        viewController action: @selector(addMaintenanceButtonTapped:)];
    UIBarButtonItem *cancelButton = [[UIBarButtonItem alloc] initWithTitle:
        NSLocalizedString(@"location.controller.navigation.cancel.title", nil) style: UIBarButtonItemStyleBordered
        target: viewController action: @selector(cancelButtonTapped:)];

    [viewController.navigationItem setLeftBarButtonItem: cancelButton];
    [viewController.navigationItem setRightBarButtonItem: addMaintenanceButton];

    // create the UITextField with equal frames
    viewController.emailField = [[SBTextField alloc] initWithFrame: CGRectMake(20, 0, 280, 44)];
    viewController.firstnameField = [[SBTextField alloc] initWithFrame: CGRectMake(20, 0, 280, 44)];
    viewController.telephone1Field = [[SBTextField alloc] initWithFrame: CGRectMake(20, 0, 280, 44)];
    viewController.telephone2Field = [[SBTextField alloc] initWithFrame: CGRectMake(20, 0, 280, 44)];

    // create address fields
    viewController.zipcodeField = [[SBTextField alloc] initWithFrame: CGRectMake(20, 0, 100, 44)];
    viewController.houseNumberField = [[SBTextField alloc] initWithFrame: CGRectMake(170, 0, 120, 44)];
    viewController.streetField = [[SBTextField alloc] initWithFrame: CGRectMake(20, 0, 280, 44)];
    viewController.townField = [[SBTextField alloc] initWithFrame: CGRectMake(20, 0, 280, 44)];

    // set vertical alignment, this is not defaulted in the iOS 6 version
    viewController.emailField.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
    viewController.firstnameField.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
    viewController.telephone1Field.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
    viewController.telephone2Field.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
    viewController.zipcodeField.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
    viewController.houseNumberField.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
    viewController.streetField.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
    viewController.townField.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter;
}

@end
```

```
//
//  SBLoginCompatabilityView_iOS_7_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBLoginCompatabilityView.h"

@interface SBLoginCompatabilityView_iOS_7_0 : SBLoginCompatabilityView

@end
```

```
//
//  SBLoginCompatabilityView_iOS_7_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBLoginCompatabilityView_iOS_7_0.h"
#import "SBPhoneLoginViewController.h"

@implementation SBLoginCompatabilityView_iOS_7_0

/*  Initialize the controller by setting the bar button and the correct frames for the textfields per version   */
- (void) initializeInterfaceOfViewController:(SBPhoneLoginViewController *)viewController
{
    // create a plus button and add it to the navigation bar
    UIBarButtonItem *addMaintenanceButton = [[UIBarButtonItem alloc] initWithTitle:
        NSLocalizedString(@"login.controller.navigation.done.title", nil) style: UIBarButtonItemStylePlain target:
        viewController action: @selector(addMaintenanceButtonTapped:)];
    UIBarButtonItem *cancelButton = [[UIBarButtonItem alloc] initWithTitle:
        NSLocalizedString(@"location.controller.navigation.cancel.title", nil) style: UIBarButtonItemStylePlain
        target: viewController action: @selector(cancelButtonTapped:)];

    [viewController.navigationItem setLeftBarButtonItem: cancelButton];
    [viewController.navigationItem setRightBarButtonItem: addMaintenanceButton];

    // create the UITextField with equal frames
    viewController.emailField = [[SBTextField alloc] initWithFrame: CGRectMake(10, 0, 300, 44)];
    viewController.firstnameField = [[SBTextField alloc] initWithFrame: CGRectMake(10, 0, 300, 44)];
    viewController.telephone1Field = [[SBTextField alloc] initWithFrame: CGRectMake(10, 0, 300, 44)];
    viewController.telephone2Field = [[SBTextField alloc] initWithFrame: CGRectMake(10, 0, 300, 44)];

    // create address fields
    viewController.zipcodeField = [[SBTextField alloc] initWithFrame: CGRectMake(10, 0, 120, 44)];
    viewController.houseNumberField = [[SBTextField alloc] initWithFrame: CGRectMake(170, 0, 120, 44)];
    viewController.streetField = [[SBTextField alloc] initWithFrame: CGRectMake(10, 0, 300, 44)];
    viewController.townField = [[SBTextField alloc] initWithFrame: CGRectMake(10, 0, 300, 44)];
}

- (void) changeProgressTitle:(NSString *)title forController:(SBPhoneLoginViewController *)viewController
{
    // just set the title of the view, no extra views are used in this version
    [viewController setTitle: title];
}

@end
```

```
//
//  SBLoginCompatabilityView.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBCompatabilityInterface.h"

@class SBPhoneLoginViewController;

@interface SBLoginCompatabilityView : NSObject <SBCompatabilityInterface>

// methods for drawing in the login view controller
- (void) drawTextField:(UITextField *) textField inTableView:(UITableView *) tableView;
- (void) initializeInterfaceOfViewController:(SBPhoneLoginViewController *)viewController;

// presents the progress view
- (void) presentProgressViewForController:(SBPhoneLoginViewController *) viewController;
- (void) changeProgressTitle:(NSString *)title forController:(SBPhoneLoginViewController *) viewController;

@end
```

```
//
//  SBLoginCompatabilityView.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 09-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBLoginCompatabilityView.h"
#import "SBPhoneLoginViewController.h"

@implementation SBLoginCompatabilityView

- (void) drawTextField:(UITextField *)textField inTableView:(UITableView *)tableView{}
- (void) initializeInterfaceOfViewController:(SBPhoneLoginViewController *)viewController{}
- (void) presentProgressViewForController:(SBPhoneLoginViewController *)viewController{}
- (void) changeProgressTitle:(NSString *)title forController:(SBPhoneLoginViewController *) viewController{}

@end
```

```
//
//  SBNavigationCompatability_iOS_6_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 28-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBNavigationCompatability.h"

@interface SBNavigationCompatability_iOS_6_0 : SBNavigationCompatability

@end
```

```
//
//  SBNavigationCompatability_iOS_6_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 28-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBNavigationCompatability_iOS_6_0.h"
#import "SBNavigationBar.h"

@implementation SBNavigationCompatability_iOS_6_0

/*  Method is called to initialize the navigation bar for the iOS 6 cocoa version   */
- (void) prepareNavigationBar:(SBNavigationBar *) navigationBar
{

}


@end
```

```
//
//  SBNavigationCompatability_iOS_7_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 28-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBNavigationCompatability.h"

@interface SBNavigationCompatability_iOS_7_0 : SBNavigationCompatability

@end
```

```
//
//  SBNavigationCompatability_iOS_7_0.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 28-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBNavigationCompatability_iOS_7_0.h"
#import "SBNavigationBar.h"

@implementation SBNavigationCompatability_iOS_7_0

/*  Prepare the iOS version navigation bar  for iOS 7   */
- (void) prepareNavigationBar:(SBNavigationBar *) navigationBar
{

}


@end
```

```
//
//  SBNavigationCompatability.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 28-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBCompatabilityInterface.h"

// predefine the navigation bar instance
@class SBNavigationBar;

@interface SBNavigationCompatability : NSObject <SBCompatabilityInterface>

// initializes the interface options specific per iOS for the current iOS version
- (void) prepareNavigationBar:(SBNavigationBar *) navigationBar;

@end
```

```
//
//  SBNavigationCompatability.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 28-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBNavigationCompatability.h"

@implementation SBNavigationCompatability

// declare method with empty implementation
- (void) prepareNavigationBar:(SBNavigationBar *)navigationBar{}

@end
```

```
//
//  SBViewCompatabilityFactory.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@class SBLocationCompatabilityView, SBCollectionCellCompatabilityView, SBLoginCompatabilityView,
    SBErrorCompatabilityView, SBImageCompatabilityView, SBKeyboardToolbarCompatabilityView,
    SBNavigationCompatability;

// define different cocoa version identifiers
typedef enum {
    UICocoaVersionIdentifier_iOS_6_0    =   0,
    UICocoaVersionIdentifier_iOS_7_0    =   1,
    UICocoaVersionIdentifier_iOS_Future =   2
} UICocoaVersionIdentifier;

@interface SBViewCompatabilityFactory : NSObject

// declare the iOS version identifier for the static factory object
@property (nonatomic) UICocoaVersionIdentifier cocoaVersionIdentifier;

// class holds strings for cached classes that can be reused
@property (nonatomic, strong) NSMutableArray *reusableClasses;

// retrieve the current cocoa version
+ (UICocoaVersionIdentifier) cocoaVersion;

// generates the class needed to make sure the UI is correct on both versions
+ (SBLocationCompatabilityView *) loadLocationCompatabilityController;
+ (SBCollectionCellCompatabilityView *) loadCompatableCollectionCell;
+ (SBLoginCompatabilityView *) loadCompatableLoginController;
+ (SBErrorCompatabilityView *) loadCompatableErrorController;
+ (SBImageCompatabilityView *) loadCompatableImageController;
+ (SBKeyboardToolbarCompatabilityView *) loadCompatableKeyboardToolbarController;
+ (Class) staticClassNameForCollectionCell;

// retrieve the compatability class object for the navigation bar
+ (SBNavigationCompatability *) navigationBarCompatability;

// method that allows per iOS version creation of bounding rect for a NSString
+ (CGSize) boundingRectForString:(NSString *) string withSize:(CGSize) size options:(NSStringDrawingOptions) options
    attributes:(NSDictionary *) attributes context:(NSStringDrawingContext *) context;

@end
```

```
//
//  SBViewCompatabilityFactory.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 08-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBViewCompatabilityFactory.h"

// location compatability view
#import "SBLocationCompatabilityView_iOS_7_0.h"
#import "SBLocationCompatabilityView_iOS_6_0.h"

// collection view cell compatability imports
#import "SBCollectionCellCompatabilityView_iOS_7_0.h"
#import "SBCollectionCellCompatabilityView_iOS_6_0.h"

// login controller draw imports
#import "SBLoginCompatabilityView_iOS_6_0.h"
#import "SBLoginCompatabilityView_iOS_7_0.h"

// error controller imports
#import "SBErrorCompatabilityView_iOS_6_0.h"
#import "SBErrorCompatabilityView_iOS_7_0.h"

// image draw controller imports
#import "SBImageCompatabilityView_iOS_6_0.h"
#import "SBImageCompatabilityView_iOS_7_0.h"

// toolbar keyboard drawing classes
#import "SBKeyboardToolbarCompatabilityView_iOS_6_0.h"
#import "SBKeyboardToolbarCompatabilityView_iOS_7_0.h"

// import files for the navigation bar
#import "SBNavigationCompatability_iOS_6_0.h"
#import "SBNavigationCompatability_iOS_7_0.h"

@interface SBViewCompatabilityFactory()

// initializes the static singleton object and set the ios version
+ (SBViewCompatabilityFactory *) sharedFactory;

// returns the reusable class when found, otherqise nil
- (id) reusableClassObjectForClass:(Class) reusableClass;

@end

@implementation SBViewCompatabilityFactory

// the singleton factory object
static SBViewCompatabilityFactory *singleton = nil;

/*  Custom initializer method for the runtime symbol check factory  */
- (SBViewCompatabilityFactory *) init
{
    self = [super init];

    if (self) {
        // set the version identifier
        self.cocoaVersionIdentifier = [SBViewCompatabilityFactory currentCocoaVersion];
        self.reusableClasses = [[NSMutableArray alloc] init];
    }

    return self;
}

/*  Returns the current cocoa version that the device is running    */
+ (UICocoaVersionIdentifier) currentCocoaVersion
{
    // set the default version that will be used
    UICocoaVersionIdentifier versionToReturn = UICocoaVersionIdentifier_iOS_6_0;

    // make the device version with the first character of the string, this will make "6", "7" etc instead of
        "6.1.4" which is too precise of a version identifier
    NSNumberFormatter *numberFormatter = [[NSNumberFormatter alloc] init];
    NSInteger deviceVersion = [[numberFormatter numberFromString: [[[UIDevice currentDevice] systemVersion]
        substringToIndex:1]] integerValue];

    // when the current version number is equal to the iOS 6.1 symbol, this device is running iOS 6.1
    if (deviceVersion <= 6) {
        versionToReturn = UICocoaVersionIdentifier_iOS_6_0;
    }
    else if (deviceVersion >= 7) {
        versionToReturn = UICocoaVersionIdentifier_iOS_7_0;
    }
```

```
    // return the found version
    return versionToReturn;
}

/*  Calculates and returns the current cocoa version the system runs on */
+ (UICocoaVersionIdentifier) cocoaVersion
{
    // return the calculated version
    return [self sharedFactory].cocoaVersionIdentifier;
}

/*  Search the array for a reusable class, must match the given parameter, when not found this method returns nil
    */
- (id) reusableClassObjectForClass:(Class) reusableClass
{
    // get the object enumerator to walk through all objects
    NSEnumerator *classEnumerator = [self.reusableClasses objectEnumerator];
    id currentObject = nil;

    // loop through the objects and try to match the object with the given class
    while (currentObject = [classEnumerator nextObject]) {
        // when the class equals to the given class return the current object
        if ([currentObject isKindOfClass: reusableClass]) {
            return currentObject;
        }
    }

    // when this part is reached nil will be returned because the class was not yet reusable
    return nil;
}

/*  creates a frame used for the given nsstring instance depending on the current iOS version   */
+ (CGSize) boundingRectForString:(NSString *) string withSize:(CGSize) size options:(NSStringDrawingOptions) options
    attributes:(NSDictionary *) attributes context:(NSStringDrawingContext *) context
{
    // the default size
    CGSize calculatedSize = CGSizeZero;

    // switch statement to calculate the correct size with available and non deprecated methods
    switch ([self sharedFactory].cocoaVersionIdentifier) {

        // when iOS 7 or future releases use the current method that should be used
        case UICocoaVersionIdentifier_iOS_7_0:
        case UICocoaVersionIdentifier_iOS_Future:
        {
            // create the rectangle from the bounding rect of the string
            CGRect calculatedFrame = [string boundingRectWithSize: size options: options attributes: attributes
                context: context];
            calculatedSize = CGSizeMake(calculatedFrame.size.width, calculatedFrame.size.height);
            break;

        }

        // the method that will be used for iOS 6
        case UICocoaVersionIdentifier_iOS_6_0:
        {
            // calculate the size
            calculatedSize = [string sizeWithFont: [attributes objectForKey: NSFontAttributeName] constrainedToSize:
                size lineBreakMode: NSLineBreakByWordWrapping];
            break;
        }
    }

    return calculatedSize;
}

/*  Return the singleton object */
+ (SBViewCompatabilityFactory *) sharedFactory
{
    // when no set create the factory
    if (singleton == nil) {
        singleton = [[SBViewCompatabilityFactory alloc] init];
    }

    return singleton;
}

/*  Load the corresponding class for the image view */
+ (SBImageCompatabilityView *) loadCompatableImageController
{
    // create default view to nil
    SBImageCompatabilityView *compatabilityView = nil;

    switch ([SBViewCompatabilityFactory sharedFactory].cocoaVersionIdentifier) {
        case UICocoaVersionIdentifier_iOS_6_0:
            compatabilityView = [[SBImageCompatabilityView_iOS_6_0 alloc] init];
            break;
```

```
            // create an object that is compatible with iOS 7
        case UICocoaVersionIdentifier_iOS_7_0:
        case UICocoaVersionIdentifier_iOS_Future:
            compatabilityView = [[SBImageCompatabilityView_iOS_7_0 alloc] init];
            break;
    }

    // return the controller
    return compatabilityView;

}

/*  Returns the class name to use for static methods    */
+ (Class) staticClassNameForCollectionCell
{
    Class className;

    switch ([SBViewCompatabilityFactory sharedFactory].cocoaVersionIdentifier) {
        case UICocoaVersionIdentifier_iOS_6_0:
            className = [SBCollectionCellCompatabilityView_iOS_6_0 class];
            break;

            // create an object that is compatible with iOS 7
        case UICocoaVersionIdentifier_iOS_7_0:
        case UICocoaVersionIdentifier_iOS_Future:
            className = [SBCollectionCellCompatabilityView_iOS_7_0 class];
            break;
    }

    return className;
}

/*  Load the corresponding controller for the current version, this is for the location selection view controller
    */
+ (SBLocationCompatabilityView *) loadLocationCompatabilityController
{
    // create default view to nil
    SBLocationCompatabilityView *compatabilityView = nil;

    switch ([SBViewCompatabilityFactory sharedFactory].cocoaVersionIdentifier) {
        case UICocoaVersionIdentifier_iOS_6_0:
            compatabilityView = [[SBLocationCompatabilityView_iOS_6_0 alloc] init];
            break;

        // create an object that is compatible with iOS 7
        case UICocoaVersionIdentifier_iOS_7_0:
        case UICocoaVersionIdentifier_iOS_Future:
            compatabilityView = [[SBLocationCompatabilityView_iOS_7_0 alloc] init];
            break;
    }

    // return the controller
    return compatabilityView;
}

/*  Load the compatable error view draw controller  */
+ (SBErrorCompatabilityView *) loadCompatableErrorController
{
    // set default compatability object
    SBErrorCompatabilityView *compatabilityView = nil;

    // get the corresponding cocoa object
    switch ([self sharedFactory].cocoaVersionIdentifier) {
        case UICocoaVersionIdentifier_iOS_6_0:
            compatabilityView = [[SBErrorCompatabilityView_iOS_6_0 alloc] init];
            break;

        case UICocoaVersionIdentifier_iOS_7_0:
        case UICocoaVersionIdentifier_iOS_Future:
            compatabilityView = [[SBErrorCompatabilityView_iOS_7_0 alloc] init];
            break;
    }

    return compatabilityView;
}

/*  Load and return the apprapriate login drawing view controller for the current version   */
+ (SBLoginCompatabilityView *) loadCompatableLoginController
{
    // default controller value
    SBLoginCompatabilityView *drawController = nil;

    // get the corresponding class
    switch ([self sharedFactory].cocoaVersionIdentifier) {
        case UICocoaVersionIdentifier_iOS_6_0:
            drawController = [[SBLoginCompatabilityView_iOS_6_0 alloc] init];
```

```
            break;

        case UICocoaVersionIdentifier_iOS_7_0:
        case UICocoaVersionIdentifier_iOS_Future:
            drawController = [[SBLoginCompatabilityView_iOS_7_0 alloc] init];
            break;
    }

    return drawController;
}

/*  Loads the class for a collection view cell, will be fetched from the reusable class when possible   */
+ (SBCollectionCellCompatabilityView *) loadCompatableCollectionCell
{
    // create default view to nil
    Class compatabilityClass = nil;

    // get the corresponding class for this cocoa version
    switch ([SBViewCompatabilityFactory sharedFactory].cocoaVersionIdentifier) {
        case UICocoaVersionIdentifier_iOS_6_0:
            compatabilityClass = [SBCollectionCellCompatabilityView_iOS_6_0 class];
            break;

        case UICocoaVersionIdentifier_iOS_7_0:
        case UICocoaVersionIdentifier_iOS_Future:
            compatabilityClass = [SBCollectionCellCompatabilityView_iOS_7_0 class];
            break;
    }

    // when a reusable object was found return it
    SBCollectionCellCompatabilityView *compatabilityObject = [singleton reusableClassObjectForClass:
        compatabilityClass];

    // check for the object to be found
    if (compatabilityObject == nil) {
        // object not found, allocate a new class and add it to the reusable array
        compatabilityObject = [[compatabilityClass alloc] init];
        [[singleton reusableClasses] addObject: compatabilityObject];
    }

    return compatabilityObject;
}

/*  Loads the class for a toolbar keyboard view   */
+ (SBKeyboardToolbarCompatabilityView *) loadCompatableKeyboardToolbarController
{
    // default controller value
    SBKeyboardToolbarCompatabilityView *drawController = nil;

    // get the corresponding class
    switch ([self sharedFactory].cocoaVersionIdentifier) {
        case UICocoaVersionIdentifier_iOS_6_0:
            drawController = [[SBKeyboardToolbarCompatabilityView_iOS_6_0 alloc] init];
            break;

        case UICocoaVersionIdentifier_iOS_7_0:
        case UICocoaVersionIdentifier_iOS_Future:
            drawController = [[SBKeyboardToolbarCompatabilityView_iOS_7_0 alloc] init];
            break;
    }

    return drawController;
}

/*  Load and create the class that will be used to initialize the navigation bar for the application    */
+ (SBNavigationCompatability *) navigationBarCompatability
{
    // create default object
    SBNavigationCompatability *initializerObject = nil;

    // depending on the current cocoa version get the current class
    switch ([self sharedFactory].cocoaVersionIdentifier) {

        case UICocoaVersionIdentifier_iOS_6_0:
            initializerObject = [[SBNavigationCompatability_iOS_6_0 alloc] init];
            break;

        // initializer for iOS 7 or higher
        case UICocoaVersionIdentifier_iOS_7_0:
        case UICocoaVersionIdentifier_iOS_Future:
            initializerObject = [[SBNavigationCompatability_iOS_7_0 alloc] init];
            break;
    }

    return initializerObject;
}
```

```
@end
```

```
//
//  SBFileOperation.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBOperation.h"
#import "SBFileOperationDelegate.h"

@interface SBFileOperation : SBOperation

@property (nonatomic, strong, readonly) NSURL *fileURL;

// initialize the file operation with a file URL for the file that will be processed
- (SBFileOperation *) initWithFileURL:(NSURL *) URL withOptions:(NSDictionary *) dictionary;

// sets the new delegate object
- (void) setDelegate:(id<SBOperationDelegate, SBFileOperationDelegate>)delegate;

@end
```

```
//
//  SBFileOperation.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBFileOperation.h"

@implementation SBFileOperation

/*  Initialize the operation with the given URL and options */
- (SBFileOperation *) initWithFileURL:(NSURL *) URL withOptions:(NSDictionary *) dictionary
{
    // init super with the specified options
    self = [super initWithOptions: dictionary];

    if (self) {

        // set the file URL
        _fileURL = URL;
    }

    return self;
}

/*  Set the new delegate object which must be a file operation delegate */
- (void) setDelegate:(id<SBOperationDelegate,SBFileOperationDelegate>)delegate
{
    [super setDelegate: delegate];
}

@end
```

```
//
//  SBFileThumbnailOperation.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBFileOperation.h"

@interface SBFileThumbnailOperation : SBFileOperation

// the image that should be processed, can be nil
@property (nonatomic, strong, readonly) UIImage *imageToProcess;

// initializes the operation to create a new thumbnail image from the specified image object
- (SBFileThumbnailOperation *) initWithImage:(UIImage *) image options:(NSDictionary *) options;

@end
```

```
//
//  SBFileThumbnailOperation.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBFileThumbnailOperation.h"
#import <ImageIO/ImageIO.h>
#import <AssetsLibrary/AssetsLibrary.h>

#define kSBImageProcessingThumbnailMaxSize      800

@interface SBFileThumbnailOperation()

// start sending delegate messages with the new created thumbnail
- (void) startSendingFileProcessedSuccessfullMessageWithImage:(UIImage *) image;

@end

@implementation SBFileThumbnailOperation

/*  Initialize the operation to prepare for the processing of an image object   */
- (SBFileThumbnailOperation *) initWithImage:(UIImage *) image options:(NSDictionary *) options
{
    // init super without URL but with options
    self = [super initWithFileURL: nil withOptions: options];

    if (self) {

        // set the private image
        _imageToProcess = image;
    }

    return self;
}

/*  Start the processing of the file to create the needed thumbnail */
- (void) startProcessing
{
    // predefine the image source
    __block CGImageSourceRef imageSource = nil;

    // when an image has been provided create the
    if (self.imageToProcess != nil) {

        // create the dataref and image source from the image JPEG representation
        CFDataRef imageDataRepresentation = CFBridgingRetain(UIImageJPEGRepresentation(self.imageToProcess, 1.0));
        imageSource = CGImageSourceCreateWithData(imageDataRepresentation, NULL);

        // release the data
        CFRelease(imageDataRepresentation);
    } else if (self.fileURL != nil) {

        // a file at the given URL should be processed
        ALAssetsLibrary *assetsLibrary = [[ALAssetsLibrary alloc] init];
        __block BOOL isLoadingAsset = YES;
        __block BOOL hasEncounteredError = NO;

        // load the image provided at the given URL from the assets library
        [assetsLibrary assetForURL: self.fileURL resultBlock: ^(ALAsset *asset) {

            // create a callback signature for the asset byte loading, is better for memory
            CGDataProviderDirectCallbacks callbacks = {
                .version = 0,
                .getBytePointer = NULL,
                .releaseBytePointer = NULL,
                .getBytesAtPosition = getAssetBytesCallback,
                .releaseInfo = releaseAssetCallback,
            };

            // get the default representation object of the asset (the full image that is)
            ALAssetRepresentation *defaultAssetRepresentation = [asset defaultRepresentation];
            CGDataProviderRef imageDataProvider = CGDataProviderCreateDirect((void *)CFBridgingRetain
                (defaultAssetRepresentation), [defaultAssetRepresentation size], &callbacks);

            // create the image source from the data provider, release the data provider when finished
            imageSource = CGImageSourceCreateWithDataProvider(imageDataProvider, NULL);
            CGDataProviderRelease(imageDataProvider);

            // loading is finished
            isLoadingAsset = NO;

        } failureBlock: ^(NSError *error) {

            // start the failing process
```

```
            hasEncounteredError = YES;
            [self failOperationWithError: error];
        }];

        // while the asset is being loaded keep running the runloop before continuing
        while (isLoadingAsset) {
            [[NSRunLoop currentRunLoop] runUntilDate: [NSDate dateWithTimeIntervalSinceNow: 0.1]];
        }

        // when an error occured during the load finish execution of the operation
        if (hasEncounteredError) {
            return;
        }
    }

    // when the image source was successfully created continue the processing
    if (imageSource != nil) {

        // notify that disk will be used
        [self willChangeValueForKey: @"isUsingDisk"];
        self.usingDisk = YES;
        [self didChangeValueForKey: @"isUsingDisk"];

        // create the options dictionary for the thumbnail creation process
        CFDictionaryRef thumbnailOptions = (__bridge_retained CFDictionaryRef)[NSDictionary
            dictionaryWithObjectsAndKeys: (id)kCFBooleanTrue, (id)kCGImageSourceCreateThumbnailWithTransform, (id)
            kCFBooleanTrue, (id)kCGImageSourceCreateThumbnailFromImageAlways, (id)[NSNumber numberWithFloat:
            kSBImageProcessingThumbnailMaxSize], (id)kCGImageSourceThumbnailMaxPixelSize, nil];

        // create the thumbnail
        CGImageRef imageThumbnail = CGImageSourceCreateThumbnailAtIndex(imageSource, 0, thumbnailOptions);

        // notify that disk usage has finished
        [self willChangeValueForKey: @"isUsingDisk"];
        self.usingDisk = NO;
        [self didChangeValueForKey: @"isUsingDisk"];

        // create the processed image object from the thimbnail data
        UIImage *image = [[UIImage alloc] initWithCGImage: imageThumbnail scale: [[UIScreen mainScreen] scale]
            orientation: UIImageOrientationUp];

        // start sending the processed successfull message and complete the operation
        [self startSendingFileProcessedSuccessfullMessageWithImage: image];
        [self completeOperation];

        // release resources
        _imageToProcess = nil;
        CFRelease(imageThumbnail);
        CFRelease(thumbnailOptions);
        CFRelease(imageSource);
    } else {

        // create info for error
        NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObject:
            NSLocalizedString(@"operation.image.thumbnail.resource.unavailable", nil)] forKeys: [NSArray
            arrayWithObject: NSLocalizedDescriptionKey]];

        // create the error and start the failing process
        NSError *error = [NSError errorWithDomain: NSCocoaErrorDomain code: -1 userInfo: userInfo];
        [self failOperationWithError: error];
    }
}

/*  Send delegate messages with the succeeded processed image   */
- (void) startSendingFileProcessedSuccessfullMessageWithImage:(UIImage *) image
{
    // when not on the main thread execute on that
    if (![[NSThread currentThread] isMainThread]) {
        [self performSelectorOnMainThread: @selector(startSendingFileProcessedSuccessfullMessageWithImage:)
            withObject: image waitUntilDone: YES];
        return;
    }

    // cast delegate as file delegate object
    id<SBOperationDelegate, SBFileOperationDelegate> object = (id<SBOperationDelegate, SBFileOperationDelegate>)self
        .delegate;

    // when the object can respond to the specified selector call it
    if ([object respondsToSelector: @selector(operation:didFinishProcessingImageWithResult:)]) {
        [object operation: self didFinishProcessingImageWithResult: image];
    }
}

#pragma mark    -
#pragma mark    C methods for asset data loading

/*  Gets called when a piece of the total bytes have been loaded from the asset image   */
```

```
static size_t getAssetBytesCallback(void *info, void *buffer, off_t position, size_t count)
{
    // get the asset object
    ALAssetRepresentation *rep = (__bridge id)info;

    // get the error and the amount of bytes that have been read
    NSError *error = nil;
    size_t countRead = [rep getBytes: (uint8_t *)buffer fromOffset: position length: count error: &error];

    // when no bytes have been read generate error
    if (countRead == 0 && error) {
        // We have no way of passing this info back to the caller, so we log it, at least.
        NSLog(@"Error reading asset: %@", error);
    }

    return countRead;
}

/*  Allows for releasing of the asset data object   */
static void releaseAssetCallback(void *info)
{
    // balance the retain and release
    CFRelease(info);
}

@end
```

```
//
//  SBImageFileSaveOperation.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBFileOperation.h"

@interface SBImageFileSaveOperation : SBFileOperation

// the image which will be saved to disk
@property (nonatomic, strong, readonly) UIImage *imageToSave;

// the optional metadata for the image that will be written to disk
@property (nonatomic, strong) NSDictionary *metadata;

// create the file save operation with the provided image
- (SBImageFileSaveOperation *) initWithImage:(UIImage *) image options:(NSDictionary *) options;

@end
```

```
//
//  SBImageFileSaveOperation.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBImageFileSaveOperation.h"
#import <AssetsLibrary/AssetsLibrary.h>

@interface SBImageFileSaveOperation()

// starts the delegate messaging with the new URL of the image file
- (void) startSendingNewFileURLMessageWithURL:(NSURL *) fileURL;

@end

@implementation SBImageFileSaveOperation

/*  Initialize the operation with the provided image and options    */
- (SBImageFileSaveOperation *) initWithImage:(UIImage *) image options:(NSDictionary *) options
{
    // create super object with no URL
    self = [super initWithFileURL: nil withOptions: options];

    if (self) {

        // set the image
        _imageToSave = image;
    }

    return self;
}

/*  Start the processing of the data    */
- (void) startProcessing
{
    // the image should be written to the documents folder in the application sandbox, get the documents folder URL
    NSURL *documentsURL = [[[NSFileManager defaultManager] URLsForDirectory: NSDocumentDirectory inDomains:
        NSUserDomainMask] lastObject];

    // create the actual URL for which the file will be written to
    NSURL *fileURL = [documentsURL URLByAppendingPathComponent: [[NSUUID UUID] UUIDString]];

    // create a data representation of the image
    NSData *fileData = UIImageJPEGRepresentation(self.imageToSave, 1.0);

    // write the file to the corresponding URL
    NSError *writeError = nil;
    if (![fileData writeToFile: [fileURL path] options: NSDataWritingAtomic error: &writeError]) {

        // fail the execution
        [self failOperationWithError: writeError];
    } else {

        // complete the operation and send new URL message
        [self startSendingNewFileURLMessageWithURL: fileURL];
        [self completeOperation];
    }
}

/*  Message the delegate objects with the new file URL of the image */
- (void) startSendingNewFileURLMessageWithURL:(NSURL *) fileURL
{
    // when not on the main thread call selector on main
    if (![[NSThread currentThread] isMainThread]) {
        [self performSelectorOnMainThread: @selector(startSendingNewFileURLMessageWithURL:) withObject: fileURL
            waitUntilDone: YES];
        return;
    }

    // cast delegate as file operation delegate
    id<SBFileOperationDelegate> object = (id<SBFileOperationDelegate>) self.delegate;

    // try to call the method on the delegate object
    if ([object respondsToSelector: @selector(operation:didFinishProcessingFileAtURL:)]) {
        [object operation: self didFinishProcessingFileAtURL: fileURL];
    }
}

@end
```

```
//
//  SBInternetCenter.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <CoreTelephony/CTTelephonyNetworkInfo.h>

// declare classes and protocols
@protocol SBInternetCenterDelegate;

// different types of connection states, the integer indicates the amount of internet operations able to be executed
    symmontamiously in current condition
typedef NS_ENUM (NSInteger, SBAppConnectionState){
    SBAppConnectionStateGPRS        =   2,
    SBAppConnectionState3G          =   5,
    SBAppConnectionStateLTE         =   12,
    SBAppConnectionStateWifi        =   15,
    SBAppConnectionStateNoInternet  =   0,
};

@interface SBInternetCenter : NSObject
{
    // the total active connections counter
    int totalActiveConnectionsCounter;

    // whether currently the indicator is shown
    BOOL isShowingActivityIndicator;

    // the network info object
    CTTelephonyNetworkInfo *networkInfo;

    // timer that is used to set racalculation
    NSTimer *recalculationtimer;
}

// a delegate object which gets notified about connection changes
@property (nonatomic, weak) id<SBInternetCenterDelegate> delegate;

// whether the connection state should be recalculated
@property (nonatomic) BOOL shouldRecalculateConnectionState;

// the current connection state
@property (nonatomic, readonly) SBAppConnectionState currentConnectionState;

// returns the shared singleton used for internet change management
+ (SBInternetCenter *) defaultCenter;

// check whether the device is connected to wifi
- (BOOL) isDeviceConnectedToWifi;

// increases the total operation counter that use outgoing connections
- (void) increaseTotalActiveConnectionCount;
- (void) decreaseTotalActiveConnectionCount;

// recalculates the current connection state for the device
- (void) recalculateDeviceConnectionState;

// start or stop the internet access state recalculation
- (void) startWifiChecker;
- (void) stopWifiChecker;

// get the total active connections
- (int) totalActiveConnections;

// whether a new internet operation is able to start with current conditions
- (BOOL) canStartInternetOperation;
@end
```

```
//
//  SBInternetCenter.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBInternetCenter.h"
#import "SBInternetCenterDelegate.h"

// headers for internet WiFi calculation
#include <ifaddrs.h>
#include <arpa/inet.h>

@interface SBInternetCenter()

// gets called when the notification center sends message with network changes
- (void) deviceRadioAccessTechnologyChangedWithNotification:(NSNotification *) notification;

@end

@implementation SBInternetCenter

// synthesize readonly variables
@synthesize currentConnectionState = _currentConnectionState;

// the shared singleton for internet center
static SBInternetCenter *sharedSingleton = nil;

/*  Initializer method that enables the notification for network information    */
- (SBInternetCenter *) init
{
    self = [super init];

    if (self) {

        // default the total active connections is 0
        totalActiveConnectionsCounter = 0;
        isShowingActivityIndicator = NO;
        _currentConnectionState = SBAppConnectionStateNoInternet;

        // create the network info obkect
        networkInfo = [[CTTelephonyNetworkInfo alloc] init];

        if ([networkInfo respondsToSelector: @selector(currentRadioAccessTechnology)]) {
            // observe the carrier network type change
            [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector
                (deviceRadioAccessTechnologyChangedWithNotification:) name:
                CTRadioAccessTechnologyDidChangeNotification object: nil];
        }
    }

    return self;
}

/*  When dealloced remove observer from notification center */
- (void) dealloc
{
    // remove observer
    [[NSNotificationCenter defaultCenter] removeObserver: self];
}

/*  Retrieve the shared internet center used for this application   */
+ (SBInternetCenter *) defaultCenter
{
    // when the default center was not initialized create it
    if (sharedSingleton == nil) {
        sharedSingleton = [[SBInternetCenter alloc] init];
    }

    // return the shared singleton
    return sharedSingleton;
}

/*  Start the Wifi connection check timer   */
- (void) startWifiChecker
{
    // check whether the timer has not already been created
    if (recalculationtimer == nil || ![recalculationtimer isValid]) {

        // create the new timer
        recalculationtimer = [NSTimer scheduledTimerWithTimeInterval: 1 target: self selector: @selector
            (recalculateDeviceConnectionState) userInfo: nil repeats: YES];

        // add the timer to the runloop
        [[NSRunLoop mainRunLoop] addTimer: recalculationtimer forMode: NSRunLoopCommonModes];
```

```
    }
}

/*  Stop the wifi checker   */
- (void) stopWifiChecker
{
    // check whether the timer is running, when true disable and dealloc the timer
    if ([recalculationtimer isValid]) {
        [recalculationtimer invalidate];
        recalculationtimer = nil;
    }
}

/*  Whether the device is connected via WiFi    */
- (BOOL) isDeviceConnectedToWifi
{
    // create the struct to hold information about the interfaces
    struct ifaddrs *currentInterfaces = NULL;
    struct ifaddrs *tempAddress = NULL;

    // whether the getting of interfaces succeeded
    int success = getifaddrs(&currentInterfaces);
    BOOL returnValue = NO;

    // when no error occured
    if (success == 0) {

        // the current addr which will be looped through
        tempAddress = currentInterfaces;

        while (tempAddress != NULL) {

            // when a IPv4 or IPv6 address is found continue
            if (tempAddress->ifa_addr->sa_family == AF_INET || tempAddress->ifa_addr->sa_family == AF_INET6) {

                // validate wether the name of the address is the interface of WiFi (en0)
                if (strcmp(tempAddress->ifa_name, "en0") == 0) {

                    // the current address is WiFi, set the return value and break out of the loop
                    returnValue = YES;
                    break;
                }
            }

            // set the next address to validate from the linked list
            tempAddress = tempAddress->ifa_next;
        }
    }

    // free up memory from interfaces and return the result
    freeifaddrs(currentInterfaces);
    return returnValue;
}

/*  Returns whether a new internet operation can be started */
- (BOOL) canStartInternetOperation
{
    NSLog(@"Can start max '%d' internet operations new value will be '%d'.", self.currentConnectionState,
        (totalActiveConnectionsCounter + 1));
    // when the new counter will enable to execution return YEs
    return ((totalActiveConnectionsCounter + 1) <= self.currentConnectionState);
}

/*  Retrieve the total active connections   */
- (int) totalActiveConnections
{
    return totalActiveConnectionsCounter;
}

/*  Starts the recalculation of the radio access technology */
- (void) recalculateDeviceConnectionState
{
    // default access technology is nil
    NSString *currentAccessTechnology = nil;

    // whether the radio access technology can be accessed
    BOOL canAccessRadioAccessTechnology = NO;
    SBAppConnectionState previousState = self.currentConnectionState;

    // check whether the network info class can respond to the radio access technology method for the version
        running on the device
    if ([networkInfo respondsToSelector: @selector(currentRadioAccessTechnology)]) {

        // create new network info object and fetch its network info
        currentAccessTechnology = [networkInfo currentRadioAccessTechnology];
        canAccessRadioAccessTechnology = YES;
    }
```

```
    // when the access technology is null this could mean a WiFi connection or no connection at all
    if (currentAccessTechnology != nil) {

        // a technology is available, first do a check whether the device is connected to WiFi so that connection
            will be used as represented
        if (![self isDeviceConnectedToWifi]) {

            // not connected to WiFi but radio access is available
            if ([currentAccessTechnology isEqualToString: CTRadioAccessTechnologyEdge] || [currentAccessTechnology
                isEqualToString: CTRadioAccessTechnologyGPRS]) {
                _currentConnectionState = SBAppConnectionStateGPRS;
            } else if ([currentAccessTechnology isEqualToString: CTRadioAccessTechnologyLTE]) {
                _currentConnectionState = SBAppConnectionStateLTE;
            } else {

                // everything else defaults to 3G
                _currentConnectionState = SBAppConnectionState3G;
            }
        } else {

            // both connected to WiFi and radio access, set Wifi as fastest internet possibility
            _currentConnectionState = SBAppConnectionStateWifi;
        }

    } else if (canAccessRadioAccessTechnology) {

        // check whether connected to WiFi
        if ([self isDeviceConnectedToWifi]) {

            // the device is only connected to WiFi
            _currentConnectionState = SBAppConnectionStateWifi;
        } else {

            // the device has no active connection with either WiFi or radio technology
            _currentConnectionState = SBAppConnectionStateNoInternet;
        }
    } else {

        // when this is reached it means that radio access technology method is not available on the device so we
            cannot say whether internet access is available. Thats why we default it to a 3G connection or wifi when
            available
        if ([self isDeviceConnectedToWifi]) {

            // set connection state to WiFi
            _currentConnectionState = SBAppConnectionStateWifi;
        } else {

            // set connection state to 3G
            _currentConnectionState = SBAppConnectionState3G;
        }
    }

    // when the connection state has changed notify delegate
    if (previousState != self.currentConnectionState) {

        // when the delegate is able to respond send the message
        if ([self.delegate respondsToSelector: @selector(internetCenter:didChangeToState:)]) {
            [self.delegate internetCenter: self didChangeToState: self.currentConnectionState];
        }
    }
}

/*  Gets called when the radio access technology changed    */
- (void) deviceRadioAccessTechnologyChangedWithNotification:(NSNotification *) notification
{
    // recalculate the values when device radio access changes
    [self recalculateDeviceConnectionState];
}

/*  Increase the total active connections count and check whether the indicator should be shown */
- (void) increaseTotalActiveConnectionCount
{
    // increase counter
    totalActiveConnectionsCounter++;

    // when the indicator is not yet shown activate it
    if (!isShowingActivityIndicator) {
        [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible: YES];
        isShowingActivityIndicator = YES;
    }
}

/*  Decrease the total active connections cound and check whether the indicator should be hidden    */
- (void) decreaseTotalActiveConnectionCount
{
    // only decrease the counter when it is higher than 0
```

```
    if (totalActiveConnectionsCounter > 0) {
        totalActiveConnectionsCounter--;
    }

    // when the total amount of connection is 0 and the indicator is visible hide it
    if (totalActiveConnectionsCounter == 0 && isShowingActivityIndicator) {
        [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible: NO];
        isShowingActivityIndicator = NO;
    }
}

@end
```

```
//
//  SBOperationNew.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

// define protocol and classes
@protocol SBOperationDelegate;
@class SBOperationStateInterface;

// keys used for the options dictionary
static NSString *SBOperationDoesRequireAuthenticationOptionKey      =
    @"SBOperationDoesRequireAuthenticationOption";
static NSString *SBOperationPriorityOptionKey                       =   @"SBOperationPriorityOptionKey";
static NSString *SBOperationIsReadyOptionKey                        =   @"SBOperationIsReadyOptionKey";
static NSString *SBOperationNameOptionKey                           =   @"SBOperationNameOptionKey";

// define state when set by the main thread but not yet responded to
typedef enum {
    SBOperationFinishedState    =   1,
    SBOperationSuspendedState   =   2,
    SBOperationUnknownState     =   3,
} SBOperationStateType;

#define kSBOperationCancelCode              -14
#define kSBOperationConnectionInvalidCode   -15

@interface SBOperation : NSOperation

// the delegate object for this operation object
@property (nonatomic, weak) id<SBOperationDelegate> delegate;

// the current state this operation is in
@property (nonatomic, weak, readonly) SBOperationStateInterface *currentState;

// options passed during the creation of the operation
@property (nonatomic, strong, readonly) NSDictionary *options;

// whether the operation is using defined resources
@property (nonatomic, getter = isUsingInternet) BOOL usingInternet;
@property (nonatomic, getter = isUsingDisk) BOOL usingDisk;

// the unique identifier for this object
@property (nonatomic, strong) NSUUID *operationIdentifier;

// defines when ready and executing and the state set by the main thread (SBOperationManager)
@property (nonatomic, getter = isReady) BOOL ready;
@property (nonatomic, getter = isExecuting) BOOL executing;

// initializes the operation abstract object with the provided options
- (SBOperation *) initWithOptions:(NSDictionary *) operationOptions;

// method that start executing in the main thread and background thread (start = "MainThread" -> main = "Background
    Thread spawned by start")
- (void) start;

// suspends the current operation and waits for later execution
- (void) suspend;

// methods that define whether the operation is executing, finished, cancelled, ready or suspended
- (BOOL) isFinished;
- (BOOL) isSuspended;

// method that allow restarting from a suspended state
- (void) restartFromSuspension;

// changes the operation to the next state object provided
- (void) changeToState:(SBOperationStateInterface *) newState;

// method that are called via the State to prepare for the next state
- (void) startProcessing;
- (void) startLoading;
- (void) cancelOperation;
- (void) failOperationWithError:(NSError *) error;
- (void) prepareForSuspension;
- (void) completeOperation;

// method indicates whether the operation will use internet
- (BOOL) willUseInternet;

// method can be implemented by subclasses to do more processing of the received API response
- (BOOL) finalizeProcessingWithError:(NSError **) error;
```

```
    // notifies the delegate object with the state change
    - (void) startSendingStateChangeMessage;

    // notifies the delegate object about the failure of this operation with the provided error object
    - (void) startSendingFailureMessageWithError:(NSError *) error;

    @end
```

```
//
//  SBOperationNew.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBOperation.h"
#import "SBOperationDelegate.h"

@interface SBOperation()
{
    // the operation type
    SBOperationStateType operationResponseState;
}

// method allows for saving of important data before suspending this operation
- (void) saveBeforeSuspension;

// the main method thatis executed in the background
- (void) main;



@end

@implementation SBOperation

// synthesize readonly properties
@synthesize options = _options;
@synthesize currentState = _currentState;

/*  Initializer method that creates an abstract operation object with the provided options  */
- (SBOperation *) initWithOptions:(NSDictionary *) operationOptions
{
    self = [super init];

    if (self) {

        // set the options
        _options = operationOptions;
        operationResponseState = SBOperationUnknownState;

        // define default states
        self.executing = NO;

        // when the options include the is ready option set the is ready value to that value
        if ([operationOptions objectForKey: SBOperationIsReadyOptionKey] != nil) {
            self.ready = [[operationOptions objectForKey: SBOperationIsReadyOptionKey] boolValue];
        } else {

            // when the ready key is not set the default is YES
            self.ready = YES;
        }
    }

    return self;
}

/*  Return YES as concurrent, the operation classes will initiate its own threads to do work on */
- (BOOL) isConcurrent
{
    return YES;
}

/*  Returns whether the operation finished with its execution   */
- (BOOL) isFinished
{
    // return whether the state is finished
    return (operationResponseState == SBOperationFinishedState);
}

/*  Returns whether this operation will eventually use internet */
- (BOOL) willUseInternet
{
    return NO;
}

/*  Whether the operation has been suspended to be executed */
- (BOOL) isSuspended
{
    // return whether the operation is cancelled
    return (operationResponseState == SBOperationSuspendedState);
}

/*  Method gets called when the operation should be suspended   */
```

```
- (void) suspend
{
    // set that the operation should be suspended
    operationResponseState = SBOperationSuspendedState;
}

/*  changes the operation to the provided state */
- (void) changeToState:(SBOperationStateInterface *) newState
{
    // set the readonly state
    _currentState = newState;

    // message the delegate with the new state change
    [self startSendingStateChangeMessage];
}

/*  Gets called when the operation can initially start its execution    */
- (void) start
{
    // check whether the operation was cancelled
    if ([self isCancelled]) {
        return;
    }

     // the operation manager should be notified that the operation starts execution
    [self willChangeValueForKey: @"isExecuting"];
    self.executing = YES;
    [self didChangeValueForKey: @"isExecuting"];

    // call the main method in a secondary thread
    [NSThread detachNewThreadSelector: @selector(main) toTarget: self withObject: nil];
}

/*  Gets called by the start method and starts the execution of the operation in the background */
- (void) main
{
    // in this implementation return directly because this method should be overridden
    [self startLoading];
    return;
}

/*  Gets called when the operation has a chance to save its state before going to suspended state   */
- (void) saveBeforeSuspension
{

}

/*  Gets called when the operation should be started from suspended state   */
- (void) restartFromSuspension
{

}

/*  Message will send the delegate message for the state change to the delegate when available  */
- (void) startSendingStateChangeMessage
{
    // only send the message on the main thred, so when not on the main thread call this method on main and return
    if (![[NSThread currentThread] isMainThread]) {
        [self performSelectorOnMainThread: @selector(startSendingStateChangeMessage) withObject: nil waitUntilDone:
            NO];
        return;
    }

    // when the delegate is able to respond to the message send it
    if ([self.delegate respondsToSelector: @selector(operation:didChangeToState:)]) {
        [self.delegate operation: self didChangeToState: self.currentState];
    }
}

/*  starts sending the delegate the failure message with a provided error object */
- (void) startSendingFailureMessageWithError:(NSError *) error
{
    // when not on the main thread the selector should be executed on the main thread
    if (![[NSThread currentThread] isMainThread]) {
        [self performSelectorOnMainThread: @selector(startSendingFailureMessageWithError:) withObject: error
            waitUntilDone: NO];
        return;
    }

    // send the delegate message when the object can respond to it
    if ([self.delegate respondsToSelector: @selector(operation:didFailWithError:)]) {
        [self.delegate operation: self didFailWithError: error];
    }
}

#pragma mark    -
#pragma mark    State changing super methods
```

```
/*  Method that should be called from subclasses to make sure the operation is actually finished    */
- (void) completeOperation
{
    // change the value of executing
    [self willChangeValueForKey: @"isExecuting"];
    self.executing = NO;
    [self didChangeValueForKey: @"isExecuting"];

    // change the value of finished
    [self willChangeValueForKey: @"isFinished"];
    operationResponseState = SBOperationFinishedState;
    [self didChangeValueForKey: @"isFinished"];
}

/*  Notifies key value observers that the operation went to a suspended state and stopped executing */
- (void) prepareForSuspension
{
    // notify that values will change
    [self willChangeValueForKey: @"isExecuting"];
    self.executing = NO;
    [self didChangeValueForKey: @"isExecuting"];

    // change to suspended state
    [self willChangeValueForKey: @"isSuspended"];
    operationResponseState = SBOperationSuspendedState;
    [self didChangeValueForKey: @"isSuspended"];
}

/*  Fails the current operation and provides the error object with the reason of failure    */
- (void) failOperationWithError:(NSError *) error
{
    // send delegate message
    [self startSendingFailureMessageWithError: error];

    // notify that values will change
    [self willChangeValueForKey: @"isExecuting"];
    self.executing = NO;
    [self didChangeValueForKey: @"isExecuting"];

    // change to finished
    [self willChangeValueForKey: @"isFinished"];
    operationResponseState = SBOperationFinishedState;
    [self didChangeValueForKey: @"isFinished"];
}

/*  Cancels the current operation and messages */
- (void) cancelOperation
{
    // create the user info dictionary with the failed message
    NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObjects:
        NSLocalizedString(@"error.operation.cancel.description", nil), nil] forKeys: [NSArray arrayWithObjects:
        NSLocalizedDescriptionKey, nil]];

    // create the error message and call the failure handler
    NSError *error = [NSError errorWithDomain: NSCocoaErrorDomain code: kSBOperationCancelCode userInfo: userInfo];
    [self failOperationWithError: error];
}

/*  Methods cnotniue to the next state immediatly because they should be implemented by the subclass objects    */
- (void) startProcessing
{
    // immediatly complete the operation
    [self completeOperation];
}

/*  Go to processing state  */
- (void) startLoading
{
    // start the processing state
    [self startProcessing];
}

/*  The default processing finalizer return YES on default, because on default no extra processing should be
    executed   */
- (BOOL) finalizeProcessingWithError:(NSError *__autoreleasing *)error
{
    return YES;
}

@end
```

```
//
//  SBOperationManager.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBInternetCenterDelegate.h"
#import "SBProcessorObserverDelegate.h"

@class SBOperation, SBInternetCenter, SBProcessorObserver;

@interface SBOperationManager : NSObject <SBInternetCenterDelegate, SBProcessorObserverDelegate>

// the queue used to schedule the operations
@property (nonatomic, strong, readonly) NSOperationQueue *operationQueue;

// the internet center that manages the current connection type and notifies about changes
@property (nonatomic, weak) SBInternetCenter *internetCenter;

// the processor observer object
@property (nonatomic, weak) SBProcessorObserver *processorObserver;

// retrieves the default operation manager used for this application
+ (SBOperationManager *) defaultManager;

// adds a new operation object to the queue to be executed, returns the unique identifier that is used to identify
    the operation, so it can be cancelled or suspended etc.
- (NSUUID *) addOperation:(SBOperation *) newOperation;

// cancel all operations with the given UUID objects in the array
- (void) cancelOperations:(NSArray *) UUIDs;

@end
```

```
//
//  SBOperationManager.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBOperationManager.h"
#import "SBOperation.h"
#import "SBInternetCenter.h"
#import "SBProcessorObserver.h"

@interface SBOperationManager()

// this method gets called when the application should stop scheduling new operations in the queue
- (void) managerShouldDisableSchedulingWithNotification:(NSNotification *) notification;

// this method gets called when the application should start scheduling its operations again
- (void) managerShouldEnableSchedulingWithNotification:(NSNotification *) notification;

@end

@implementation SBOperationManager

// synthesize the read only operation queue
@synthesize operationQueue = _operationQueue;

// the static singleton used in the entire application
static SBOperationManager *sharedSingleton = nil;

/*  Creates a new operation manager object with an empty queue  */
- (SBOperationManager *) init
{
    self = [super init];

    if (self) {

        // create the operation queue to use for all operations
        _operationQueue = [[NSOperationQueue alloc] init];
        [_operationQueue addObserver: self forKeyPath: @"operationCount" options: NSKeyValueObservingOptionNew
            context: NULL];

        // set internet center and delegate object
        self.internetCenter = [SBInternetCenter defaultCenter];
        [self.internetCenter setDelegate: self];

        // set the processor observer and delegate
        self.processorObserver = [SBProcessorObserver defaultObserver];
        [self.processorObserver setDelegate: self];
    }

    return self;
}

/*  Retrieves the default manager used in the application   */
+ (SBOperationManager *) defaultManager
{
    // when the object has not been created to that now
    if (sharedSingleton == nil) {
        sharedSingleton = [[SBOperationManager alloc] init];
    }

    // return the singleton object
    return sharedSingleton;
}

/*  Adds a new operation to the queue waiting to be executed    */
- (NSUUID *) addOperation:(SBOperation *) newOperation
{
    // start listening to some of the variables when changed
    [newOperation addObserver: self forKeyPath: @"isUsingInternet" options: NSKeyValueObservingOptionNew context:
        NULL];
    [newOperation addObserver: self forKeyPath: @"isUsingDisk" options: NSKeyValueObservingOptionNew context: NULL];
    [newOperation addObserver: self forKeyPath: @"isReady" options: NSKeyValueObservingOptionNew context: NULL];
    [newOperation addObserver: self forKeyPath: @"isExecuting" options: NSKeyValueObservingOptionNew context: NULL];
    [newOperation addObserver: self forKeyPath: @"isFinished" options: NSKeyValueObservingOptionNew context: NULL];
    [newOperation addObserver: self forKeyPath: @"isSuspended" options: NSKeyValueObservingOptionNew context: NULL];

    // set the unique identifier of the operation
    NSUUID *uniqueIdentifier = [NSUUID UUID];
    [newOperation setOperationIdentifier: uniqueIdentifier];

    // add operation to queue and return
    [self.operationQueue addOperation: newOperation];
    return uniqueIdentifier;
}
```

```
/*  Cancels all operations found with the UUIDs provided in the Array   */
- (void) cancelOperations:(NSArray *) UUIDs
{
    // create the enumerator for the NSUUID objects
    NSEnumerator *objectEnumerator = [UUIDs objectEnumerator];
    NSUUID *currentIdentifier = nil;

    // loop through all UUIDs
    while (currentIdentifier = [objectEnumerator nextObject]) {

        // enumerate through all available operations
        [[self.operationQueue operations] enumerateObjectsUsingBlock: ^(SBOperation *operation, NSUInteger idx, BOOL
            *stop) {

            // when the UUId is equal to the current operations identifier cancel it
            if ([[operation operationIdentifier] isEqual: currentIdentifier]) {

                // when this operation is not cancelled cancel it now
                if (![operation isCancelled]) {
                    [operation cancel];
                }

                // tell the enumerator to stop
                *stop = YES;
            }
        }];
    }
}


/*  Listen to key-value changes for a operation object and respond to that change   */
- (void) observeValueForKeyPath:(NSString *) keyPath ofObject:(id) object change:(NSDictionary *) change context:
    (void *) context
{
    // all changes should be new boolean values when received
    BOOL newValue = [[change objectForKey: NSKeyValueChangeNewKey] boolValue];

    // when the changed keypath is "isFinished" check whether the keyvalue observer should be released, this should
        happend when the value is set to YES
    if ([keyPath isEqualToString: @"isFinished"] && newValue) {

        // release all observers for the provided keypaths because the operation shall be dealloced
        [object removeObserver: self forKeyPath: @"isFinished"];
        [object removeObserver: self forKeyPath: @"isUsingInternet"];
        [object removeObserver: self forKeyPath: @"isUsingDisk"];
        [object removeObserver: self forKeyPath: @"isReady"];
        [object removeObserver: self forKeyPath: @"isExecuting"];
        [object removeObserver: self forKeyPath: @"isSuspended"];

    } else if ([keyPath isEqualToString: @"isUsingInternet"]) {

        // increase or decrease the activity counter depending on the new value
        if (newValue) {
            [self.internetCenter increaseTotalActiveConnectionCount];
        } else {
            [self.internetCenter decreaseTotalActiveConnectionCount];
        }
    }
}

#pragma mark    -
#pragma mark    Scheduling methods

/*  Gets notified when the application should disable scheduling because the application will enter background etc.
    */
- (void) managerShouldDisableSchedulingWithNotification:(NSNotification *) notification
{
    // disable scheduling
    [self.operationQueue setSuspended: YES];
}

/*  Gets notified when the application should start the scheduling again, i.e. the application enters foreground
    */
- (void) managerShouldEnableSchedulingWithNotification:(NSNotification *) notification
{
    // reenable scheduling
    [self.operationQueue setSuspended: NO];
}

#pragma mark    -
#pragma mark    Internet & Processor change delegation methods

/*  When the internet center detects a change in radio access technology changes respond using this method   */
- (void) internetCenter:(SBInternetCenter *) center didChangeToState:(NSInteger) newState
{
    NSLog(@"New connection state: '%d'.", newState);
```

```
}

/*  When the processor state changes this method gets called with the new percentage    */
- (void) processorObserver:(SBProcessorObserver *) observer didObserveProcessorUsage:(float) usage
{
    NSLog(@"Current processor usage: %f", [observer inactivePercentage]);
}

@end
```

```
//
//  SBProcessorObserver.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 27-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "SBProcessorObserverDelegate.h"
#include <mach/processor_info.h>

// defines the unknown activity processing percentage
#define kSBProcessorObserverUnknownActivity     0

@interface SBProcessorObserver : NSObject
{
    // indicates the number of processors available on the system (the number of threads the can be concurrently
       executed)
    int numberOfProcessors;

    // the total percentage of all cores combined that is in use, from a scale of 0 to 1
    float usedProcessorPercentage;

    // the timer that will be used to keep recalculating the processor usage
    NSTimer *observationTimer;

    // the current and previous processor info structs
    processor_info_array_t currentProcessorInfo;
    processor_info_array_t previousProcessorInfo;

    // the mach type information for the current and previous
    mach_msg_type_number_t currentProcessorNumberInfo;
    mach_msg_type_number_t previousProcessorNumberInfo;
};

// the delegate object that receives notifications about processing changes
@property (nonatomic, strong) id<SBProcessorObserverDelegate> delegate;

// retrieve the shared observer object
+ (SBProcessorObserver *) defaultObserver;

// get information about the current processing system and activity
- (int) coreCount;
- (float) activityPercentage;
- (float) inactivePercentage;

// start the processing observation process
- (void) startMonitoring;
- (void) stopMonitoring;

@end
```

```
//
//  SBProcessorObserver.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 27-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBProcessorObserver.h"
// include C libraries for processor usage calculation
#include <sys/sysctl.h>
//#include <sys/types.h>
#include <mach/mach.h>
#include <mach/processor_info.h>
#include <mach/mach_host.h>

@interface SBProcessorObserver()

// starts the cycle that recalculates the processing activity at this moment
- (void) recalculateProcessorActivity;

// calculate the in use activity percentage for the given core
- (float) totalActivityForCore:(unsigned int) coreNumber;

@end

@implementation SBProcessorObserver

// the default observing singleton
static SBProcessorObserver *sharedSingleton = nil;

/*  Creates the processor observer and initializes default values   */
- (SBProcessorObserver *) init
{
    self = [super init];

    if (self) {

        // by default the number of processors is 1, because the application is run on iPhones
        int tempProcessorCount = 1;

        // variables for fetching the number of processing units / Cores, the CTL_HW indicates processor / IO
            fetching and the HW_NCPU indicates that we want to fetch the number of processing cores (sync threads)
        int mib[2U] = {CTL_HW, HW_NCPU};
        size_t sizeOfNumberOfProcessors = sizeof(tempProcessorCount);
        int status = sysctl(mib, 2U, &tempProcessorCount, &sizeOfNumberOfProcessors, NULL, 0U);

        // when status is no success set number of processor units to 1
        if (status != 0) {
            NSLog(@"Unable to fetch the number of cores, setting default number of cores.");
        }

        // set the number of processors and the default activity count
        numberOfProcessors = tempProcessorCount;
        usedProcessorPercentage = kSBProcessorObserverUnknownActivity;
    }

    return self;
}

/*  Retrieve the default observer singleton */
+ (SBProcessorObserver *) defaultObserver
{
    // when not created initialize the singleton
    if (sharedSingleton == nil) {
        sharedSingleton = [[SBProcessorObserver alloc] init];
    }

    return sharedSingleton;
}

/*  Retrieve the amount of cores for this system    */
- (int) coreCount
{
    return numberOfProcessors;
}

/*  Retrieve the current activity percentage    */
- (float) activityPercentage
{
    return (usedProcessorPercentage * 100);
}

/*  Retrieve the current free percentage    */
- (float) inactivePercentage
{
    return (100 - [self activityPercentage]);
```

```
}

/*  Start the actitvity monitoring timer and process    */
- (void) startMonitoring
{
    // do initial calculation to notify immediatly
    [self recalculateProcessorActivity];

    // when the timer is not already running create and activate it
    if (observationTimer == nil || ![observationTimer isValid]) {

        // create the timer which recalculates every 0.1 seconds
        observationTimer = [NSTimer scheduledTimerWithTimeInterval: 0.5 target: self selector: @selector
            (recalculateProcessorActivity) userInfo: nil repeats: YES];

        // add the timer to the main runloop
        [[NSRunLoop mainRunLoop] addTimer: observationTimer forMode: NSRunLoopCommonModes];
    }
}

/*  Stops the current activity monitoring process and timer */
- (void) stopMonitoring
{
    // when the timer is not already disabled invalidate it
    if (observationTimer != nil || [observationTimer isValid]) {

        // invalidate and set to nil
        [observationTimer invalidate];
        observationTimer = nil;
    }
}

/*  Recalculate the processing activity for all cores that are available    */
- (void) recalculateProcessorActivity
{
    // set default number of processor and load the processor info in the property "currentProcessorInfo", this
        value will contain core information and activity that we will process when successfully fetched
    natural_t numProcessorsU = 0U;
    kern_return_t error = host_processor_info(mach_host_self(), PROCESSOR_CPU_LOAD_INFO,  &numProcessorsU, &
        currentProcessorInfo, &currentProcessorNumberInfo);

    // when no error occured during the fetching of the cpu load continue
    if (error == KERN_SUCCESS) {

        // set the default usage
        float currentPercentage = kSBProcessorObserverUnknownActivity;

        // loop through each processing core unit
        for (unsigned int i = 0U; i < numberOfProcessors; i++) {

            // calculate the usage for the current core, add this result to the total activity
            currentPercentage += [self totalActivityForCore: i];
        }

        // calculate the total usage using the data of all the cores
        usedProcessorPercentage = (currentPercentage / (numberOfProcessors * 100)) * 100;

        // when the previous processor info struct was set deallocate it from virtual memory
        if (previousProcessorInfo) {

            // calculate the size of the previous processor info with the number of the previous info
            size_t previousProcessorInfoSize = sizeof(integer_t) * previousProcessorNumberInfo;

            // deallocate virtual memory for the previous processor info in this mach task
            vm_deallocate(mach_task_self(), (vm_address_t)previousProcessorInfo, previousProcessorInfoSize);
        }

        // replace the previous processor info values with the current values
        previousProcessorInfo = currentProcessorInfo;
        previousProcessorNumberInfo = currentProcessorNumberInfo;

        // reset for next calculation
        currentProcessorInfo = NULL;
        numProcessorsU = 0U;

    } else {

        // set the activity to unknown
        usedProcessorPercentage = kSBProcessorObserverUnknownActivity;
    }

    // when a delegate object has been set notify this object with the new activity float
    if ([self.delegate respondsToSelector: @selector(processorObserver:didObserveProcessorUsage:)]) {
        [self.delegate processorObserver: self didObserveProcessorUsage: (usedProcessorPercentage * 100)];
    }
}
```

```
/*  Get the total activity for the core number provided */
- (float) totalActivityForCore:(unsigned int) coreNumber
{
    // define in use and the total usage
    float processorCurrentUsage = 0;
    float processorUsageTotal = 0;

    // when a previous value is set
    if (previousProcessorInfo) {

        // get the processor usage of the following types: (USER, SYSTEM & NICE). With this variables we can
            calculate the system performance at the moment
        processorCurrentUsage = ((currentProcessorInfo[(CPU_STATE_MAX * coreNumber) + CPU_STATE_USER] -
            previousProcessorInfo[(CPU_STATE_MAX * coreNumber) + CPU_STATE_USER]) + (currentProcessorInfo
            [(CPU_STATE_MAX * coreNumber) + CPU_STATE_SYSTEM] - previousProcessorInfo[(CPU_STATE_MAX * coreNumber) +
            CPU_STATE_SYSTEM]) + (currentProcessorInfo[(CPU_STATE_MAX * coreNumber) + CPU_STATE_NICE] -
            previousProcessorInfo[(CPU_STATE_MAX * coreNumber) + CPU_STATE_NICE]));

        // calculate the total usage
        processorUsageTotal = processorCurrentUsage + (currentProcessorInfo[(CPU_STATE_MAX * coreNumber) +
            CPU_STATE_IDLE] - previousProcessorInfo[(CPU_STATE_MAX * coreNumber) + CPU_STATE_IDLE]);
    }
    else {
        // calculate the current in usa of the processor
        processorCurrentUsage = currentProcessorInfo[(CPU_STATE_MAX * coreNumber) + CPU_STATE_USER] +
            currentProcessorInfo[(CPU_STATE_MAX * coreNumber) + CPU_STATE_SYSTEM] + currentProcessorInfo
            [(CPU_STATE_MAX * coreNumber) + CPU_STATE_NICE];

        // calculate total usage
        processorUsageTotal = processorCurrentUsage + currentProcessorInfo[(CPU_STATE_MAX * coreNumber) +
            CPU_STATE_IDLE];
    }

    // return the calculated activity for the core that should be calculated
    return (processorCurrentUsage / processorUsageTotal);
}

@end
```

```
//
//  SBURLCachedImageOperation.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBURLOperation.h"

@interface SBURLCachedImageOperation : SBURLOperation
{
    // private file manager for file read and write
    NSFileManager *fileManager;

    // whether the loaded image should be cached
    BOOL shouldCacheDownloadedImage;

    // the new filename for the objects cached file
    NSString *fileName;
}

// the readonly managed object ID for which the image should be loaded
@property (nonatomic, strong, readonly) NSManagedObjectID *objectID;

// initialize the cached image operation with the provided managed object ID and options
- (SBURLCachedImageOperation *) initWithManagedObjectID:(NSManagedObjectID *) managedObjectID withOptions:
    (NSDictionary *) options;

@end
```

```
//
//  SBURLCachedImageOperation.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBURLCachedImageOperation.h"
#import "SBManagedObjectInterface.h"
#import "SBAppDelegate.h"
#import "SBURLOperationDelegate.h"
#import "SBAPIRequest.h"
#import "SBAPIResponse.h"

#define kSBAPICachedImageTimeout        2678400

@interface SBURLCachedImageOperation()

// prepares the file system for cached files, returns whether cached files can be used
+ (BOOL) hasCacheDirectoryAccess;

// determines whether the file is available for the managed object ID provided, sets the image when found
- (BOOL) isCachedFileAvailableWithName:(NSString *) fileName result:(NSData **) cachedData;

// saves the cached image to disk for later reference
- (void) saveReceivedDataToCacheForFileName:(NSString *) fileName;

// prepare for sending delegate methods with the provided image
- (void) startSendingImageLoadedMessagesWithImage:(UIImage *) image;

@end

@implementation SBURLCachedImageOperation

// synthesize readonly properties
@synthesize objectID = _objectID;

// the directory name for downloaded images
static NSString *SBAPICachedImagesDirectoryName     =   @"Images";

// static variables indicating cache is available
static BOOL cacheCheckIsExecuted = NO;
static BOOL isCachedDirectoryAvailableAndWriteable = NO;

/*  Initialize the operation with the provided managed object ID and options    */
- (SBURLCachedImageOperation *) initWithManagedObjectID:(NSManagedObjectID *) managedObjectID withOptions:
    (NSDictionary *) options
{
    // initialize super with no request but with the options
    self = [super initWithAPIRequest: nil options: options];

    if (self) {

        // set readonly managed object ID
        _objectID = managedObjectID;
        shouldCacheDownloadedImage = NO;

        // create the file manager
        fileManager = [NSFileManager defaultManager];
    }

    return self;
}

/*  Starts the loading process of the image cache operation */
- (void) startLoading
{
    // whether the image was loaded from cache
    BOOL isImageLoadedFromCache = NO;

    // create a new managed object context for this background thread
    SBAppDelegate *applicationDelegate = (SBAppDelegate *)[[UIApplication sharedApplication] delegate];
    NSManagedObjectContext *managedObjectContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:
        NSConfinementConcurrencyType];
    [managedObjectContext setPersistentStoreCoordinator: [applicationDelegate persistentStoreCoordinator]];

    // fetch the existing object from the context store
    NSManagedObject<SBManagedObjectInterface> *managedObject = (NSManagedObject <SBManagedObjectInterface>*)
        [managedObjectContext objectWithID: self.objectID];

    // when the managed object culd not be fetched create error
    if (managedObject == nil) {

        // create userinfo dictionary
        NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObject:
            NSLocalizedString(@"operation.cache.image.object.unavailable", nil)] forKeys: [NSArray arrayWithObject:
```

```
            NSLocalizedDescriptionKey]];

        // create error object with provided userinfo
        NSError *error = [NSError errorWithDomain: NSCocoaErrorDomain code: -1 userInfo: userInfo];
        [self failOperationWithError: error];
        return;
    }

    // create the filename for the entity
    fileName = [NSString stringWithFormat: @"%@%@.png", [[managedObject entity] name], [managedObject
        externalObjectID]];

    // check whether the application has read and write access to the defined cache directory
    if ([SBURLCachedImageOperation hasCacheDirectoryAccess]) {

        // create reference object in which the image will be placed
        NSData *cachedData = nil;

        // check whether the image is already cached
        if ([self isCachedFileAvailableWithName: fileName result: &cachedData]) {

            // set the URL response to enable usage of the base class parsing method
            NSURLResponse *response = [[NSURLResponse alloc] initWithURL: nil MIMEType: @"image/png"
                expectedContentLength: [cachedData length] textEncodingName: nil];
            _APIResponse = [[SBAPIResponse alloc] initWithURLResponse: response];
            _downloadedData = [NSData dataWithBytes: [cachedData bytes] length: [cachedData length]];

            // complete the operation, in this method the image will be parsed and returned to the delegate
            [self startProcessing];
            isImageLoadedFromCache = YES;
        }
    }

    // only load the image when it was not loaded from cache
    if (!isImageLoadedFromCache) {

        // create the API request, because these images should be able to load fairly quickly the timeout for this
            request is 10 seconds
        SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPIGetImageRequestType timeout: 10];

        // configure the request
        [APIRequest setPOSTValue: [[managedObject entity] name] forKey: @"type"];
        [APIRequest setPOSTValue: [managedObject externalObjectID] forKey: @"id"];

        // create the url connection object
        self.URLConnection = [[NSURLConnection alloc] initWithRequest: [APIRequest finalURLRequest] delegate: self
            startImmediately: NO];

        // create the url connection for this object
        if (self.URLConnection != nil) {

            // start the connection
            [self.URLConnection start];

            // notify observer that internet activity has begun
            [self willChangeValueForKey: @"isUsingInternet"];
            self.usingInternet = YES;
            [self didChangeValueForKey: @"isUsingInternet"];

            // keep looping until the data has been loaded
            while (![self isCancelled]) {

                // when loading is complete the data can be parsed
                if (self.isLoadingComplete) {
                    break;
                }

                // run the loop for 0.25 seconds
                [[NSRunLoop currentRunLoop] runUntilDate: [NSDate dateWithTimeIntervalSinceNow: 0.25]];
            }

            // notify observers that internet access has finished for the operation
            [self willChangeValueForKey: @"isUsingInternet"];
            self.usingInternet = NO;
            [self didChangeValueForKey: @"isUsingInternet"];

            // when the operation is cancelled stop execution
            if (![self isCancelled]) {

                // start the processing of the data
                shouldCacheDownloadedImage = [_APIResponse isImage];
                [self startProcessing];

                // when an image is loaded notify delegate
                if (shouldCacheDownloadedImage) {
                    [self startSendingImageLoadedMessagesWithImage: _APIResponse.imageResource];
                } else {
```

```
                    // create info for the error
                    NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObject:
                        NSLocalizedString(@"error.operation.no.image.resource", nil)] forKeys: [NSArray
                        arrayWithObject: NSLocalizedDescriptionKey]];

                    // create error and start failing process
                    NSError *error = [NSError errorWithDomain: NSCocoaErrorDomain code: -1 userInfo: userInfo];
                    [self failOperationWithError: error];
                }

            } else {

                // create info for the error
                NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObject:
                    NSLocalizedString(@"error.operation.cancel.description", nil)] forKeys: [NSArray
                    arrayWithObject: NSLocalizedDescriptionKey]];

                // create error and start failing process
                NSError *error = [NSError errorWithDomain: NSCocoaErrorDomain code: -1 userInfo: userInfo];
                [self failOperationWithError: error];
            }
        } else {

            // create userinfo because the connection could not be created
            NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObject:
                NSLocalizedString(@"error.operation.request.invalid.description", nil)] forKeys: [NSArray
                arrayWithObject: NSLocalizedDescriptionKey]];

            // create error and start the failing process
            NSError *error = [NSError errorWithDomain: NSCocoaErrorDomain code: -1 userInfo: userInfo];
            [self failOperationWithError: error];
        }
    } else {

        // notify user with the image
        [self startSendingImageLoadedMessagesWithImage: _APIResponse.imageResource];
    }
}

/*  Finalize the processing error   */
- (BOOL) finalizeProcessingWithError:(NSError *__autoreleasing *) error
{
    // check whether the image should and can be cached
    if (shouldCacheDownloadedImage && [SBURLCachedImageOperation hasCacheDirectoryAccess]) {

        // save the downloaded data to cache
        [self saveReceivedDataToCacheForFileName: fileName];
    }

    return YES;
}

/*  Send the cached image result to the delegate    */
- (void) startSendingImageLoadedMessagesWithImage:(UIImage *)image
{
    // when not on the main thread recall on main thread
    if (![[NSThread currentThread] isMainThread]) {
        [self performSelectorOnMainThread: @selector(startSendingImageLoadedMessagesWithImage:) withObject: image
            waitUntilDone: YES];
        return;
    }

    NSLog(@"Image: %@", image);

    // cast the delegat as url operation delegate
    id<SBOperationDelegate, SBURLOperationDelegate> object = (id<SBOperationDelegate, SBURLOperationDelegate>) self.
        delegate;

    // message delegate when able to respond
    if ([object respondsToSelector: @selector(operation:didFinishWithImage:)]) {
        [object operation: self didFinishWithImage: image];
    }
}

/*  Determines whether the application can cache images to the directory for this usage */
+ (BOOL) hasCacheDirectoryAccess
{
    // when already defined whether the cached files directory is available return the previous result
    if (cacheCheckIsExecuted) {
        return isCachedDirectoryAvailableAndWriteable;
    }

    // first we have to check whether the image is already available in the downloaded images cache
    NSFileManager *defaultManager = [NSFileManager defaultManager];
    NSURL *cachesURL = [[defaultManager URLsForDirectory: NSCachesDirectory inDomains: NSUserDomainMask] lastObject]
        ;
```

```
    // boolean will be set when the asked file is a directory
    BOOL isDirectoryAtPath = NO;
    BOOL canSaveToDirectory = NO;

    // check whether the cache directory is available
    if (![defaultManager fileExistsAtPath: [[cachesURL URLByAppendingPathComponent: SBAPICachedImagesDirectoryName]
        path] isDirectory: &isDirectoryAtPath]) {

        // try to create the directory
        if ([defaultManager createDirectoryAtURL: [cachesURL URLByAppendingPathComponent:
            SBAPICachedImagesDirectoryName] withIntermediateDirectories: NO attributes: nil error: nil]) {

            // directory is successfully created
            canSaveToDirectory = YES;
        }
    } else if (isDirectoryAtPath) {

        // the found file exists and is a directory, check whether this directory is writable
        if ([defaultManager isWritableFileAtPath: [[cachesURL URLByAppendingPathComponent:
            SBAPICachedImagesDirectoryName] path]]) {
            canSaveToDirectory = YES;
        }
    }

    // set the static variables
    cacheCheckIsExecuted = YES;
    isCachedDirectoryAvailableAndWriteable = canSaveToDirectory;

    // return whether save can be performed
    return isCachedDirectoryAvailableAndWriteable;
}

/*  Saves the downloaded bytes to a new file in the cache directory for further reference   */
- (void) saveReceivedDataToCacheForFileName:(NSString *) file
{
    // get the object URI to fetch the unique ID of this managed object, the host will get the UUID string of this
        URL
    NSURL *cacheDirectoryPath = [[[fileManager URLsForDirectory: NSCachesDirectory inDomains: NSUserDomainMask]
        lastObject] URLByAppendingPathComponent: SBAPICachedImagesDirectoryName isDirectory: YES];

    // the final URL of the image
    NSURL *fileURL = [cacheDirectoryPath URLByAppendingPathComponent: file];

    // save the downloaded data to disk, do not mind with errors for a cached file
    [_downloadedData writeToURL: fileURL options: NSDataWritingAtomic error: nil];
}

/*  Returns the cache image when available for the given object ID or else returns NO   */
- (BOOL) isCachedFileAvailableWithName:(NSString *) file result:(NSData *__autoreleasing *) cachedData
{
    // create the URL to the cached directory
    NSURL *cacheDirectoryPath = [[[fileManager URLsForDirectory: NSCachesDirectory inDomains: NSUserDomainMask]
        lastObject] URLByAppendingPathComponent: SBAPICachedImagesDirectoryName isDirectory: YES];

    // the final URL of the image
    NSURL *fileURL = [cacheDirectoryPath URLByAppendingPathComponent: file];
    BOOL fileIsFound = NO;

    // when the file exists and i readable for the application continue and read the data
    if ([fileManager fileExistsAtPath: [fileURL path] isDirectory: NULL] && [fileManager isReadableFileAtPath:
        [fileURL path]]) {

        // check for the age of the file
        NSError *attributeError = nil;
        NSDictionary *fileAttributes = [fileManager attributesOfItemAtPath: [fileURL path] error: &attributeError];

        // when no error has occured read the file attributes data and check for the cached file to still be valid
        if (attributeError == nil) {

            // get the creation date of the file
            NSDate *fileCreationDate = [fileAttributes objectForKey: NSFileCreationDate];

            // get time intervals for now ad the creation date
            NSTimeInterval timeInterval = -[fileCreationDate timeIntervalSinceNow];

            // when the file is expired delete it from cache
            if (timeInterval > kSBAPICachedImageTimeout) {

                // delete the file from the cache directory
                NSError *deletionError = nil;
                if (![fileManager removeItemAtPath: [fileURL path] error: &deletionError]) {
                    NSLog(@"Error deleting cache file: %@", deletionError);
                }

                // image should be reloaded, return NO so the image is being downloaded
                return NO;
```

```
            }
        }

        // notify that the appication will start reading data from file system
        [self willChangeValueForKey: @"isUsingDisk"];
        self.usingDisk = YES;
        [self didChangeValueForKey: @"isUsingDisk"];

        // the file exists and is readable, get the data of this file
        NSData *fileData = [fileManager contentsAtPath: [fileURL path]];

        // notify observer that file access has finished
        [self willChangeValueForKey: @"isUsingDisk"];
        self.usingDisk = NO;
        [self didChangeValueForKey: @"isUsingDisk"];

        // when data is read create the image object
        if (fileData != nil) {

            // set the data object and return true because the image data was successfully loaded
            *cachedData = fileData;
            fileIsFound = YES;
        }
    }

    // return whether the cache data is found
    return fileIsFound;
}

@end
```

```
//
//  SBURLOperation.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBOperation.h"
#import "SBURLOperationDelegate.h"

@class SBAPIRequest;

// option key that defines whether the operation should notify about progress changes
static NSString *SBOperationNotifyProgressOptionKey     =   @"SBOperationNotifyProgressOptionKey";

@interface SBURLOperation : SBOperation <NSURLConnectionDelegate, NSURLConnectionDataDelegate>
{
    @protected

    // set protected properties that are available for subclasses to be accessed
    SBAPIResponse *_APIResponse;
    NSMutableData *_downloadedData;
}
// the expected content length of the resource that is being loaded and the current total loaded content length
@property (nonatomic) NSInteger expectedContentLength;
@property (nonatomic) NSInteger loadedContentLength;

// the container for the downloaded bytes
@property (nonatomic, strong, readonly) NSMutableData *downloadedData;

// the request from which the resource should be downloaded
@property (nonatomic, strong, readonly) SBAPIRequest *APIRequest;
@property (nonatomic, strong, readonly) SBAPIResponse *APIResponse;

// the connection object which is used to send and load the appropriate data
@property (nonatomic, strong) NSURLConnection *URLConnection;
@property (nonatomic, getter = isLoadingComplete) BOOL loadingComplete;

// whether the operation should notify the delegate with byte upload and download progress
@property (nonatomic) BOOL shouldNotifyProgress;

// initializes the operation with the provided API request and options
- (SBURLOperation *) initWithAPIRequest:(SBAPIRequest *) request options:(NSDictionary *) options;

// sets the new delegate object
- (void) setDelegate:(id<SBOperationDelegate, SBURLOperationDelegate>) delegate;

@end
```

```
//
//  SBURLOperation.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBURLOperation.h"
#import "SBAPIResponse.h"
#import "SBAPIRequest.h"
#import "SBURLOperationDelegate.h"

@interface SBURLOperation()

// messages the delegate with the retrieved API response object
- (void) startSendingOperationCompleteMessage;

@end

@implementation SBURLOperation

// synthesize readonly properties
@synthesize downloadedData = _downloadedData;
@synthesize APIRequest = _APIRequest;

// Initialize the new url operation with the provided request object and options dictionary *
- (SBURLOperation *) initWithAPIRequest:(SBAPIRequest *) request options:(NSDictionary *) options
{
    // initialize super
    self = [super initWithOptions: options];

    if (self) {

        // set the request object
        _APIRequest = request;
        self.loadingComplete = NO;

        // when the options key for progress notifications have been set
        if ([options objectForKey: SBOperationNotifyProgressOptionKey] != nil) {
            self.shouldNotifyProgress = [[options objectForKey: SBOperationNotifyProgressOptionKey] boolValue];
        } else {

            // default these notification should not be sent
            self.shouldNotifyProgress = NO;
        }
    }

    return self;
}

/*  Prepare for being deallocated   */
- (void) dealloc
{

}

/*  Sets the new delegate object which should be a URLOperation delegate    */
- (void) setDelegate:(id<SBOperationDelegate,SBURLOperationDelegate>) delegate
{
    // set the super delegate object
    [super setDelegate: delegate];
}

/*  Override the main method which should implement the basic operation execution process   */
- (void) main
{
    // when not cancelled the operation can start to be executed
    if (![self isCancelled]) {

        // intitialize by starting the loading process
        [self startLoading];
    } else {

        // cancel the operation
        [self cancelOperation];
    }
}

/*  Start to process the received data to the API response object, this object can be an image or JSON object   */
- (void) startProcessing
{
    // create objects that will be set by the APIRequest
    NSError *parseError = nil;

    // when the data should be an image parse the image object
    if (![self.APIResponse parseData: self.downloadedData withError: &parseError]) {
```

```
        // an error occured during the parsing of the downloaded data, prepare for failure
        [self failOperationWithError: parseError];
    } else {

        // do custom parsing of the data
        if ([self finalizeProcessingWithError: &parseError]) {

            // completed processing
            [self completeOperation];
        } else {

            // fail the operation with the rovided error
            [self failOperationWithError: parseError];
        }
    }
}

/*  Starts the loading part of the operation    */
- (void) startLoading
{
    // create the connection object from the URL request
    self.loadingComplete = NO;
    self.URLConnection = [[NSURLConnection alloc] initWithRequest: [self.APIRequest finalURLRequest] delegate: self
        startImmediately: NO];

    // when the _URLConnection was successfully created the operation can start loading the content
    if (self.URLConnection != nil) {

        // notify key value observers that internet activity is activated
        [self willChangeValueForKey: @"isUsingInternet"];
        self.usingInternet = YES;
        [self didChangeValueForKey: @"isUsingInternet"];

        // start the loading process
        [self.URLConnection start];

        // keep running the loop until the data has been loaded
        while (![self isCancelled]) {

            // break out of the loop when done loading
            if ([self isLoadingComplete]) {
                break;
            }

            // run the runloop and retry after 0.25 seconds
            [[NSRunLoop currentRunLoop] runUntilDate: [NSDate dateWithTimeIntervalSinceNow: 0.25]];
        }

        // notify key value observers that internet activity is deactivated
        [self willChangeValueForKey: @"isUsingInternet"];
        self.usingInternet = NO;
        [self didChangeValueForKey: @"isUsingInternet"];

        // when cancelled the operation should fail
        if ([self isCancelled]) {

            // cancel the operation
            [self cancelOperation];
        } else {

            // go to the processing stats
            [self startProcessing];
        }
    } else {

        // create the user info dictionary with the failed message
        NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObjects:
            NSLocalizedString(@"error.operation.request.invalid.description", nil), [self.APIRequest URL], nil]
            forKeys: [NSArray arrayWithObjects: NSLocalizedDescriptionKey, NSURLErrorKey, nil]];

        // create error object and change the state to failed
        NSError *error = [NSError errorWithDomain: NSCocoaErrorDomain code: kSBOperationConnectionInvalidCode
            userInfo: userInfo];
        [self failOperationWithError: error];
    }
}

/*  this operation will use internet    */
- (BOOL) willUseInternet
{
    return YES;
}

/*  Cancels the current operation, in this case the connection is cancelled */
- (void) cancelOperation
{
```

```
    // cancels the current connection object when not already done
    if (self.URLConnection != nil) {
        [self.URLConnection cancel];
        self.URLConnection = nil;
    }

    // reset the received data to release it
    _downloadedData = nil;

    // call super implementation
    [super cancelOperation];
}

/*  This method should handle suspension and should cancel the operation execution and save its current state   */
- (void) prepareForSuspension
{
    // method will be used for Proof of Concept

    // call super implementation
    [super prepareForSuspension];
}

/*  Message finishes the execution of the operation and messages the delegate that the operation finished execution
    */
- (void) completeOperation
{
    // start sending the delegate message
    [self startSendingOperationCompleteMessage];

    // call super implementation
    [super completeOperation];
}

/*  Starts sending the delegate object the message that the operation completed execution with the provided API
    response     object */
- (void) startSendingOperationCompleteMessage
{
    // when no delegate is set return immediatly
    if (self.delegate == nil) {
        return;
    }

    // execute this method on the main thread when not already, in this case we want to wait until the execution
        finishes or else the API result object can be dealloced
    if (![[NSThread currentThread] isMainThread]) {
        [self performSelectorOnMainThread: @selector(startSendingOperationCompleteMessage) withObject: nil
            waitUntilDone: YES];
        return;
    }

    // cast the delegate as URLOperation delegate
    id<SBOperationDelegate, SBURLOperationDelegate> object = (id<SBOperationDelegate, SBURLOperationDelegate>) self.
        delegate;

    // call finished method when the object can respond to that
    if ([object respondsToSelector: @selector(operation:didFinishWithResult:)]) {
        [object operation: self didFinishWithResult: self.APIResponse];
    }
}

#pragma mark    -
#pragma mark    NSURLConnection delegate methods

/*  Gets called when the operation failed to be executed    */
- (void) connection:(NSURLConnection *) connection didFailWithError:(NSError *) error
{
    // fail the operation with the provided error
    [self failOperationWithError: error];
    self.loadingComplete = YES;
}

/*  Gets called when the connection receives an authentication challenge before accessing specific content  */
- (void) connection:(NSURLConnection *) connection didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *
    ) challenge
{

}

/*  Gets called when new data is available to be written to the downloaded data container   */
- (void) connection:(NSURLConnection *) connection didReceiveData:(NSData *) data
{
    // append the received data to the current container
    [self.downloadedData appendData: data];
}

/*  Gets called when the connection receives a HTTP response, this method should reset the data container   */
- (void) connection:(NSURLConnection *) connection didReceiveResponse:(NSURLResponse *) response
```

```
{
    // create the response object with the HTTP URL response received from the server
    _APIResponse = [[SBAPIResponse alloc] initWithURLResponse: response];

    // reset the data container
    _downloadedData = [NSMutableData dataWithLength: 0];
}

/*  Gets called when the connection finished downloading its content successfully   */
- (void) connectionDidFinishLoading:(NSURLConnection *) connection
{
    NSString *str = [[NSString alloc] initWithData: self.downloadedData encoding: NSUTF8StringEncoding];
    NSLog(@"%@", str);

    // sets that the loading has completed and processing should begin
    self.loadingComplete = YES;
}

@end
```

```
//
//  SBURLReservationOperation.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 24-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBURLOperation.h"

@class Reservation;

@interface SBURLReservationOperation : SBURLOperation
{
    //an array of API requests that will be uploaded
    NSMutableArray *requestTasks;

    // defines whether the user information is uploaded
    BOOL isUserUploaded;

    // defines whether the operation can continue to upload the next maintenance
    BOOL shouldContinueToNextMaintenance;

    // whether an error already has occured and has been processed in the state changes
    BOOL errorIsGenerated;
}

// the reservation that will be uploaded
@property (nonatomic, strong, readonly) Reservation *reservation;

// the error that occured during the upload
@property (nonatomic, strong) NSError *error;

// custom creates the operation with a reservation object
- (SBURLReservationOperation *) initWithReservation:(Reservation *) reservation options:(NSDictionary *) options;

@end
```

```
//
//  SBURLReservationOperation.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 24-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBURLReservationOperation.h"
#import "SBAPIRequest.h"
#import "SBAPIResponse.h"
#import "Reservation.h"
#import "Proceeding.h"
#import "SBUser.h"
#import "SBDayPartDescriptor.h"
#import "Defect.h"

@interface SBURLReservationOperation()

// individualy parses the JSON data received from the uploaded / downloaded content
- (BOOL) parseJSONForCurrentResponse:(NSError **) error;

@end

// the user id identifier for the JSON result dictionary when a user was uploaded
static NSString *SBAPIUserUploadResultingIDKey      =   @"user_id";

@implementation SBURLReservationOperation

// for the readonly reservation property
@synthesize reservation = _reservation;

/*  Creates a new reservation upload operation with the provided reservation and options    */
- (SBURLReservationOperation *) initWithReservation:(Reservation *) reservation options:(NSDictionary *) options
{
    // init super object with only the options
    self = [super initWithAPIRequest: nil options: options];

    if (self) {

        // set the reservation object that needs to be uploaded
        _reservation = reservation;
        isUserUploaded = NO;
        errorIsGenerated = NO;
        shouldContinueToNextMaintenance = NO;

        // initialize the task array, initial capacity is one because the user will be first uploaded
        requestTasks = [NSMutableArray arrayWithCapacity: 1];
    }

    return self;
}

/*  Starts one of the loading processes of the reservation upload process   */
- (void) startLoading
{
    // set loading as incomplete
    self.loadingComplete = NO;

    // when the user has not been uploaded start the upload request for the current user
    if (!isUserUploaded) {

        // create the API request object that uploads the user
        SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPIAddUserRequestType timeout: 120];

        // configure the user request by adding all required values
        [APIRequest setPOSTValue: [[self.reservation reservationUser] userFirstName] forKey: @"name"];
        [APIRequest setPOSTValue: [[self.reservation reservationUser] userEmail] forKey: @"email"];
        [APIRequest setPOSTValue: [[self.reservation reservationUser] userTelephoneNumber] forKey: @"phone_number"];
        [APIRequest setPOSTValue: [[self.reservation reservationUser] zipcode] forKey: @"zipcode"];
        [APIRequest setPOSTValue: [[self.reservation reservationUser] houseNumber] forKey: @"house_number"];
        [APIRequest setPOSTValue: [[self.reservation reservationUser] addition] forKey: @"number_addition"];
        [APIRequest setPOSTValue: [[[self.reservation reservationUser] userCustomer] customerId] forKey:
            @"customer_id"];
        [APIRequest setPOSTValue: [NSString stringWithFormat: @"%f", [[self.reservation selectedDate]
            timeIntervalSince1970]] forKey: @"date"];
        [APIRequest setPOSTValue: [[self.reservation daypart] timeId] forKey: @"daypart_id"];

        // set optional values
        if ([[[self.reservation reservationUser] userOptionalTelephoneNumber] length] > 0) {
            [APIRequest setPOSTValue: [[self.reservation reservationUser] userOptionalTelephoneNumber] forKey:
                @"phone_number_optional"];
        }

        // set the notification token when set
        if ([[self.reservation reservationUser] deviceToken] != nil) {
            [APIRequest setPOSTValue: [[self.reservation reservationUser] deviceToken] forKey: @"notification_token"
```

```
                ];
        }

        // add the API request to the requests array that need to be uploaded
        [requestTasks addObject: APIRequest];
    }

    // loop through all the requests that need to be uploaded
    for (int i = 0; i < [requestTasks count]; i++) {

        // create the connection object for the current API request
        self.URLConnection = [[NSURLConnection alloc] initWithRequest: [[requestTasks objectAtIndex: i]
            finalURLRequest] delegate: self startImmediately: NO];

        // when the connection was successfully created start the upload
        if (self.URLConnection != nil) {

            // start the connection
            [self.URLConnection start];

            // notify observers that the internet will be accessed
            [self willChangeValueForKey: @"isUsingInternet"];
            self.usingInternet = YES;
            [self didChangeValueForKey: @"isUsingInternet"];

            // keep looping until the operation is cancelled
            while (![self isCancelled]) {

                // check whether the operation can continue to the next upload
                if (shouldContinueToNextMaintenance) {
                    break;
                }

                // run the loop for 0.25 seconds
                [[NSRunLoop currentRunLoop] runUntilDate: [NSDate dateWithTimeIntervalSinceNow: 0.25]];
            }

            // notify observers that internet finished accessing
            [self willChangeValueForKey: @"isUsingInternet"];
            self.usingInternet = NO;
            [self didChangeValueForKey: @"isUsingInternet"];

            // when error is generated fail the operation
            if ([self isCancelled]) {

                // cancel the operation and break out of the loop
                [self cancelOperation];
                return;
            } else if (!errorIsGenerated) {

                // reset that the next maitenance has not been uploaded, when there is a next object available
                shouldContinueToNextMaintenance = NO;
            }
        } else {

            // create user info for the error
            NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObject:
                NSLocalizedString(@"error.operation.request.invalid.description", nil)] forKeys: [NSArray
                arrayWithObject: NSLocalizedDescriptionKey]];

            // mark the operation as failed and break to finish the loading process
            NSError *error = [NSError errorWithDomain: NSCocoaErrorDomain code: -1 userInfo: userInfo];
            [self failOperationWithError: error];
            return;
        }
    }

    // when no error has occured the operation can be completed, else an error has already been notified
    if (!errorIsGenerated) {

        // set the new status of the reservation
        [self.reservation setCurrentStatus: [NSNumber numberWithInt: SBReservationStatusWaiting]];
        [self completeOperation];
    } else {

        // set failed state
        [self.reservation setCurrentStatus: [NSNumber numberWithInt: SBReservationStatusFailed]];
        [self failOperationWithError: self.error];
    }
}

/*  Parses the JSON data received for the current connection and returns whether the content is valid   */
- (BOOL) parseJSONForCurrentResponse:(NSError *__autoreleasing *)error
{
    // return whether data processing has been successfull
    return [self.APIResponse parseData: self.downloadedData withError: error];
}
```

```
#pragma mark    -
#pragma mark    URLConnection override methods

/*  When a connection finished loading its contents check whether to add the defect tasks to the request array  */
- (void) connectionDidFinishLoading:(NSURLConnection *) connection
{
    NSString *str = [[NSString alloc] initWithData: _downloadedData encoding: NSUTF8StringEncoding];
    NSLog(@"%@", str);

    // create error that will be set during parsing when it fails
    NSError *parseError = nil;

    // try to parse the JSON data
    if ([self parseJSONForCurrentResponse: &parseError]) {

        // when the user was not yet uploaded this means that the data received contains the result of the user
            upload process
        if (!isUserUploaded) {

            // get the user id of the user that was uploaded
            NSNumber *userId = [[self.APIResponse resultDictionary] objectForKey: SBAPIUserUploadResultingIDKey];

            // when this user id is set create the other API requests for this reservation
            if (userId != nil) {

                // set the external id for the reservation
                [self.reservation setExternalReservationId: userId];

                // get the proceeding enumerator for the reservation
                NSEnumerator *proceedingEnumerator = [[self.reservation proceedings] objectEnumerator];
                Proceeding *currentProceeding = nil;

                // add all maintenance (Proceeding) requests to the task array
                while (currentProceeding = [proceedingEnumerator nextObject]) {

                    // create a new request for adding a new maintenance
                    SBAPIRequest *newAPIRequest = [[SBAPIRequest alloc] initWithRequestCall:
                        SBAPIAddMaintenanceRequestType timeout: 120];

                    // add required POST values
                    [newAPIRequest setPOSTValue: [currentProceeding proceedingDescription] forKey: @"description"];
                    [newAPIRequest setPOSTValue: [[currentProceeding defect] defectId]  forKey: @"defect_id"];
                    [newAPIRequest setPOSTValue: userId forKey: @"user_id"];

                    // when this is the last maintenance that will be uploaded mark the maintenance as the last
                    if ([[proceedingEnumerator allObjects] count] == 0) {
                        [newAPIRequest setPOSTValue: [NSNumber numberWithBool: YES] forKey: @"is_final_maintenance"]
                            ;
                    }

                    // add the image when available
                    if ([currentProceeding imageDescriptor] != nil) {
                        [newAPIRequest addPOSTFileData: nil withFileName: @"maintenance_image.jpg" MIMEType:
                            @"image/jpeg"];
                    }

                    // add the request to the enumerator
                    [requestTasks addObject: newAPIRequest];
                    shouldContinueToNextMaintenance = YES;
                    isUserUploaded = YES;
                }
            } else {

                // create user info for the operation failed error
                NSDictionary *userInfo = [NSDictionary dictionaryWithObjects: [NSArray arrayWithObjects:
                    NSLocalizedString(@"operation.reservation.upload.failed.user", nil), nil] forKeys: [NSArray
                    arrayWithObjects: NSLocalizedDescriptionKey, nil]];

                // create error and mark the operation as failed
                self.error = [NSError errorWithDomain: NSCocoaErrorDomain code: -1 userInfo: userInfo];
                self.loadingComplete = YES;
                shouldContinueToNextMaintenance = YES;
                errorIsGenerated = YES;
            }

        } else {

            // an indivudual defect object has been uploaded
            shouldContinueToNextMaintenance = YES;
        }

    } else {

        // mark the operation as failed
        self.loadingComplete = YES;
        errorIsGenerated = YES;
```

```
            [self failOperationWithError: parseError];
        }
    }

    @end
```

```
//
//  SBURLSynchronizeOperation.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 24-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBURLOperation.h"

@interface SBURLSynchronizeOperation : SBURLOperation
{
    // the number formatter that is used for formatting ID objects
    NSNumberFormatter *numberFormatter;
}

// determines whether the application should start synchronizing the defect choices
+ (BOOL) needsSynchronization;

@end
```

```
//
//  SBURLSynchronizeOperation.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 24-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBURLSynchronizeOperation.h"
#import "SBAPIResponse.h"
#import "SBAppDelegate.h"
#import "SBRelationDescriptor.h"

// managed objects
#import "Element.h"
#import "Space.h"
#import "Defect.h"
#import "Location.h"

@interface SBURLSynchronizeOperation()

// deletes all existing location objects which also deletes all underlying objects
- (void) deleteExistingLocationsInManagedObjectContext:(NSManagedObjectContext *) managedObjectContext;

// processes all received locations in the provided array in the managed object context, returns false and sets the
    error object when unsuccesfull
- (BOOL) processLocations:(NSArray *) locations inManagedObjectContext:(NSManagedObjectContext *)
    managedObjectContext withError:(NSError **) error;

// processes all received spaces in the provided array in the managed object context, returns false and sets the
    error object when unsuccesfull
- (BOOL) processSpaces:(NSArray *) spaces inManagedObjectContext:(NSManagedObjectContext *) managedObjectContext
    withError:(NSError **) error;

// processes all received elements in the provided array in the managed object context, returns false and sets the
    error object when unsuccesfull
- (BOOL) processElements:(NSArray *) elements inManagedObjectContext:(NSManagedObjectContext *) managedObjectContext
    withError:(NSError **) error;

// processes all received defects in the provided array in the managed object context, returns false and sets the
    error object when unsuccesfull
- (BOOL) processDefects:(NSArray *) defects inManagedObjectContext:(NSManagedObjectContext *) managedObjectContext
    withError:(NSError **) error;

@end

// strings that identify the arrays of objects that can be updated
static NSString *SBAPILocationSynchronizationKey        =   @"locations";
static NSString *SBAPISpacesSynchronizationKey          =   @"spaces";
static NSString *SBAPIElementSynchronizationKey         =   @"elements";
static NSString *SBAPIDefectSynchronizationKey          =   @"defects";

// string used to identify property names in individual dictionaries
static NSString *SBAPISynchronizeDescriptionPropertyKey =   @"description";
static NSString *SBAPISynchronizeIDPropertyKey          =   @"id";

// values for the individual sync objects
static NSString *SBAPISynchronizeElementIDPropertyKey   =   @"om_element_id";
static NSString *SBAPISynchronizeSpaceIDPropertyKey     =   @"om_space_id";
static NSString *SBAPISynchronizeLocationIDPropertyKey  =   @"om_location_id";

// the user defaults string which states the previous date of synchronization
static NSString *SBAPISynchronizationDateKey            =   @"SBAPISynchronizationDateKey";

// define the amount of seconds it takes before synchronization occurs (1 week)
#define kSBAPISynchronizationTimeout    604800

@implementation SBURLSynchronizeOperation

// statis boolean which defines whether the application is already busy synchronizing
static BOOL isSynchronizing = NO;

/*  Determines whether the application refresh should be started    */
+ (BOOL) needsSynchronization
{
    // when already synchronizing return NO
    if (isSynchronizing) {
        return NO;
    }

    // try to fetch the previous date from the user defaults
    NSDate *previousSynchronizationDate = [[NSUserDefaults standardUserDefaults] objectForKey:
        SBAPISynchronizationDateKey];

    // when no date is specified the application should refresh
    if (previousSynchronizationDate == nil) {
        return YES;
```

```
    } else {

        // get the current date interval since 1970
        NSTimeInterval nowInterval = [NSDate timeIntervalSinceReferenceDate];
        NSTimeInterval syncInterval = [previousSynchronizationDate timeIntervalSinceReferenceDate];

        // when the difference is greater than the max allowed timeout the application should refresh, otherwise not
        return ((nowInterval - syncInterval) > kSBAPISynchronizationTimeout);
    }
}

/*  Override main method to indicate that the operations sync has started   */
- (void) start
{
    // set synchronizing to YES and call super implementation
    isSynchronizing = YES;
    [super start];
}

/*  Override the complete method to indicate syncing is finished    */
- (void) completeOperation
{
    isSynchronizing = NO;
    [super completeOperation];
}

/*  Override fail implementation to handel resetting to not synchronizing   */
- (void) failOperationWithError:(NSError *) error
{
    isSynchronizing = NO;
    [super failOperationWithError: error];
}

/*  When the processing of the received data can be started this method should be executed  */
- (BOOL) finalizeProcessingWithError:(NSError *__autoreleasing *) error
{
    // retrieve the data dictionary that should be processed to the CoreData database
    NSDictionary *objects = [self.APIResponse resultDictionary];

    // create and configure the number formatter
    numberFormatter = [[NSNumberFormatter alloc] init];
    NSError *processError = nil;

    // get the application delegate object to retrieve the main and persistent store coordinator
    SBAppDelegate *applicationDelegate = (SBAppDelegate *)[[UIApplication sharedApplication] delegate];

    // create a new managed object context for the current thread and set its stor coordinator
    NSManagedObjectContext *managedObjectContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:
        NSConfinementConcurrencyType];
    [managedObjectContext setPersistentStoreCoordinator: [applicationDelegate persistentStoreCoordinator]];

    // lock the context to make sure only this thread has access to its content
    [managedObjectContext lock];

    // the first step is to delete all existing locations for a clean defect database
    [self deleteExistingLocationsInManagedObjectContext: managedObjectContext];

    // process the locations array
    if ([objects objectForKey: SBAPILocationSynchronizationKey] != nil && ![self processLocations: [objects
        objectForKey: SBAPILocationSynchronizationKey] inManagedObjectContext: managedObjectContext withError: &
        processError]) {

        // on failure immediatly return
        *error = processError;
        return NO;
    }

    // process the locations array
    if ([objects objectForKey: SBAPISpacesSynchronizationKey] != nil && ![self processSpaces: [objects objectForKey:
        SBAPISpacesSynchronizationKey] inManagedObjectContext: managedObjectContext withError: &processError]) {

        // on failure immediatly return
        *error = processError;
        return NO;
    }

    // process the locations array
    if ([objects objectForKey: SBAPIElementSynchronizationKey] != nil && ![self processElements: [objects
        objectForKey: SBAPIElementSynchronizationKey] inManagedObjectContext: managedObjectContext withError: &
        processError]) {

        // on failure immediatly return
        *error = processError;
        return NO;
    }

    // process the locations array
```

```
    if ([objects objectForKey: SBAPIDefectSynchronizationKey] != nil && ![self processDefects: [objects
        objectForKey: SBAPIDefectSynchronizationKey] inManagedObjectContext: managedObjectContext withError: &
        processError]) {

        // on failure immediatly return
        *error = processError;
        return NO;
    }

    // notify observers that the disk will be used
    [self willChangeValueForKey: @"isUsingDisk"];
    self.usingDisk = YES;
    [self didChangeValueForKey: @"isUsingDisk"];

    // when there are changes in the current managed object context and no error occurs during saving
    if ([managedObjectContext hasChanges] && ![managedObjectContext save: error]) {
        return NO;
    }

    // notify observers that the disk has been used
    [self willChangeValueForKey: @"isUsingDisk"];
    self.usingDisk = NO;
    [self didChangeValueForKey: @"isUsingDisk"];

    // set the new synchronization date
    [[NSUserDefaults standardUserDefaults] setObject: [NSDate date] forKey: SBAPISynchronizationDateKey];

    // unlock the managed object context, processing the data has finished and the finalization has completed
    [managedObjectContext unlock];
    return YES;
}

/*  Deletes all current location objects from the store */
- (void) deleteExistingLocationsInManagedObjectContext:(NSManagedObjectContext *)managedObjectContext
{
    // fetch all existing locations and delete them
    NSFetchRequest *locationRequest = [[NSFetchRequest alloc] init];
    [locationRequest setEntity: [NSEntityDescription entityForName: @"Location" inManagedObjectContext:
        managedObjectContext]];

    // when set the properties wont be fetched which is quicker to load
    [locationRequest setIncludesPropertyValues: NO];

    // perform the fetch request
    NSError *error = nil;
    NSArray *locations = [managedObjectContext executeFetchRequest: locationRequest error: &error];

    // create enumerator
    NSEnumerator *locationEnumerator = [locations objectEnumerator];
    NSManagedObject *currentObject = nil;

    // delete all objects in the enumerated array
    while (currentObject = [locationEnumerator nextObject]) {
        [managedObjectContext deleteObject: currentObject];
    }
}

/*  Insert all new location objects in the CoreData store for the background thread context */
- (BOOL) processLocations:(NSArray *)locations inManagedObjectContext:(NSManagedObjectContext *)managedObjectContext
    withError:(NSError *__autoreleasing *)error
{
    // create the element entity description
    NSEntityDescription *locationDescriptor = [NSEntityDescription entityForName: @"Location"
        inManagedObjectContext: managedObjectContext];

    // create enumerator for the element objects
    NSEnumerator *elementEnumerator = [locations objectEnumerator];
    NSDictionary *currentElementInfo = nil;

    // process each info dictionary to the context
    while (currentElementInfo = [elementEnumerator nextObject]) {

        // create a new Element object from the provided element description object and insert it in the managed
            object context of this thread
        Location *newLocation = [[Location alloc] initWithEntity: locationDescriptor insertIntoManagedObjectContext:
            managedObjectContext];

        // set the properties of this new element
        [newLocation setLocationId: [numberFormatter numberFromString: [currentElementInfo objectForKey:
            SBAPISynchronizeIDPropertyKey]]];
        [newLocation setLocationDescription: [currentElementInfo objectForKey:
            SBAPISynchronizeDescriptionPropertyKey]];
    }

    return YES;
}
```

```
/*  Process all space objectes in the array to the CoreData store for the background context thread */
- (BOOL) processSpaces:(NSArray *) spaces inManagedObjectContext:(NSManagedObjectContext *) managedObjectContext
    withError:(NSError *__autoreleasing *) error
{
    // fetch all space objects for relationship coupling
    NSError *fetchError = nil;
    NSFetchRequest *spaceRequest = [NSFetchRequest fetchRequestWithEntityName: @"Location"];
    NSArray *relationshipLocations = [managedObjectContext executeFetchRequest: spaceRequest error: &fetchError];

    // stop executing when the error was set
    if (fetchError != nil) {
        *error = fetchError;
        return NO;
    }

    // create the element entity description
    NSEntityDescription *spaceDescriptor = [NSEntityDescription entityForName: @"Space" inManagedObjectContext:
        managedObjectContext];

    // create enumerator for the element objects
    NSEnumerator *elementEnumerator = [spaces objectEnumerator];
    NSDictionary *currentElementInfo = nil;
    Location *currentLocation = nil;

    // process each info dictionary to the context
    while (currentElementInfo = [elementEnumerator nextObject]) {

        // create a new Element object from the provided element description object and insert it in the managed
            object context of this thread
        Space *newSpace = [[Space alloc] initWithEntity: spaceDescriptor insertIntoManagedObjectContext:
            managedObjectContext];

        // set the properties of this new element
        [newSpace setSpaceId: [numberFormatter numberFromString: [currentElementInfo objectForKey:
            SBAPISynchronizeIDPropertyKey]]];
        [newSpace setSpaceDescription: [currentElementInfo objectForKey: SBAPISynchronizeDescriptionPropertyKey]];
        NSNumber *locationId = [numberFormatter numberFromString: [currentElementInfo objectForKey:
            SBAPISynchronizeLocationIDPropertyKey]];

        // create new enumerator
        NSEnumerator *relationshipEnumerator = [relationshipLocations objectEnumerator];

        // find the space object with the provided id, this object should be added this object as coupled
        while (currentLocation = [relationshipEnumerator nextObject]) {

            // when the relationship id matches process the relationship and go to the next object
            if ([[currentLocation locationId] isEqualToNumber: locationId]) {

                // set the coupled space
                [newSpace setLocation: currentLocation];
                break;
            }
        }
    }

    return YES;
}

/*  Process all element objects in the array to the CoreData store for the background context thread    */
- (BOOL) processElements:(NSArray *) elements inManagedObjectContext:(NSManagedObjectContext *) managedObjectContext
    withError:(NSError *__autoreleasing *) error
{
    // fetch all space objects for relationship coupling
    NSError *fetchError = nil;
    NSFetchRequest *spaceRequest = [NSFetchRequest fetchRequestWithEntityName: @"Space"];
    NSArray *relationshipSpaces = [managedObjectContext executeFetchRequest: spaceRequest error: &fetchError];

    // stop executing when the error was set
    if (fetchError != nil) {
        *error = fetchError;
        return NO;
    }

    // create the element entity description
    NSEntityDescription *elementDescriptor = [NSEntityDescription entityForName: @"Element" inManagedObjectContext:
        managedObjectContext];

    // create enumerator for the element objects
    NSEnumerator *elementEnumerator = [elements objectEnumerator];
    NSDictionary *currentElementInfo = nil;
    Space *currentSpace = nil;

    // process each info dictionary to the context
    while (currentElementInfo = [elementEnumerator nextObject]) {

        // create a new Element object from the provided element description object and insert it in the managed
            object context of this thread
```

```
        Element *newElement = [[Element alloc] initWithEntity: elementDescriptor insertIntoManagedObjectContext:
            managedObjectContext];

        // set the properties of this new element
        [newElement setElementId: [numberFormatter numberFromString: [currentElementInfo objectForKey:
            SBAPISynchronizeIDPropertyKey]]];
        [newElement setElementDescription: [currentElementInfo objectForKey: SBAPISynchronizeDescriptionPropertyKey]
            ];
        NSNumber *spaceId = [numberFormatter numberFromString: [currentElementInfo objectForKey:
            SBAPISynchronizeSpaceIDPropertyKey]];

        // create new enumerator
        NSEnumerator *relationshipEnumerator = [relationshipSpaces objectEnumerator];

        // find the space object with the provided id, this object should be added this object as coupled
        while (currentSpace = [relationshipEnumerator nextObject]) {

            // when the relationship id matches process the relationship and go to the next object
            if ([[currentSpace spaceId] isEqualToNumber: spaceId]) {

                // set the coupled space
                [newElement setSpace: currentSpace];
                break;
            }
        }
    }

    return YES;
}

/*  Processes all defects to the CoreData store in the background managed object context    */
- (BOOL) processDefects:(NSArray *) defects inManagedObjectContext:(NSManagedObjectContext *) managedObjectContext
    withError:(NSError *__autoreleasing *) error
{
    // fetch all space objects for relationship coupling
    NSError *fetchError = nil;
    NSFetchRequest *elementRequest = [NSFetchRequest fetchRequestWithEntityName: @"Element"];
    NSArray *relationshipDefects = [managedObjectContext executeFetchRequest: elementRequest error: &fetchError];

    // stop executing when the error was set
    if (fetchError != nil) {
        *error = fetchError;
        return NO;
    }

    // create the element entity description
    NSEntityDescription *defectDescriptor = [NSEntityDescription entityForName: @"Defect" inManagedObjectContext:
        managedObjectContext];

    // create enumerator for the element objects
    NSEnumerator *defectEnumerator = [defects objectEnumerator];
    NSDictionary *currentDefectInfo = nil;
    Element *currentElement = nil;

    // process each info dictionary to the context
    while (currentDefectInfo = [defectEnumerator nextObject]) {

        // create a new Element object from the provided element description object and insert it in the managed
            object context of this thread
        Defect *newDefect = [[Defect alloc] initWithEntity: defectDescriptor insertIntoManagedObjectContext:
            managedObjectContext];

        // set the properties of this new element
        [newDefect setDefectId: [numberFormatter numberFromString: [currentDefectInfo objectForKey:
            SBAPISynchronizeIDPropertyKey]]];
        [newDefect setDefectDescription: [currentDefectInfo objectForKey: SBAPISynchronizeDescriptionPropertyKey]];
        NSNumber *elementId = [numberFormatter numberFromString: [currentDefectInfo objectForKey:
            SBAPISynchronizeElementIDPropertyKey]];

        // create enumerator
        NSEnumerator *relationshipEnumerator = [relationshipDefects objectEnumerator];

        // find the space object with the provided id, this object should be added this object as coupled
        while (currentElement = [relationshipEnumerator nextObject]) {

            // when the relationship id matches process the relationship and go to the next object
            if ([[currentElement elementId] isEqualToNumber: elementId]) {
                // set the coupled space
                [newDefect setElement: currentElement];
                break;
            }
        }
    }

    return YES;
}
```

```
@end
```

```
//
//  SBCollectionControllerViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 19-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBManagedObjectInterface.h"
#import "SBCollectionControllerDelegate.h"
#import "SBOperationDelegate.h"
#import "SBURLOperationDelegate.h"

@interface SBCollectionController : UIView <NSFetchedResultsControllerDelegate, UICollectionViewDataSource,
    UICollectionViewDelegate, SBOperationDelegate, SBURLOperationDelegate>

// the collection view for this controller
@property (nonatomic, strong) UICollectionView *collectionView;
@property (nonatomic, strong) NSFetchRequest *collectionFetchRequest;
@property (nonatomic, strong) NSFetchedResultsController *collectionFetchedResultsController;
@property (nonatomic, strong) UIActivityIndicatorView *activityIndicator;
@property (nonatomic, strong) NSMutableArray *collectionAnimationObjects;
@property (nonatomic, strong) NSIndexPath *indexPathOfSelectedObject;
@property (nonatomic, weak) id<SBCollectionControllerDelegate> delegate;

// initializer method
- (SBCollectionController *) initWithFrame:(CGRect) frame fetchRequest:(NSFetchRequest *) request;
- (void) reset;

// when the controller should move to a next view controller this method needs to be overloaded
- (NSNumber *) identifierForSelectedIndexPath;

// retreive the managed object that is selected from the controller
- (NSManagedObject *) managedObjectForSelectedIndexPath;

@end
```

```
//
//  SBCollectionControllerViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 19-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBCollectionController.h"
#import "SBAppDelegate.h"
#import "SBFetchedResultsControllerAnimation.h"
#import "SBElementCollectionViewCell.h"
#import "SBCollectionCellViewLayout.h"
#import "SBCollectionViewCellTitleView.h"
#import "SBURLCachedImageOperation.h"
#import "SBOperationManager.h"
#import "SBAPIResponse.h"

// define identifiers for supplementary views and a collection view cell
static NSString * const SBCollectionCellIdentifier      = @"PhotoCell";
static NSString * const SBCollectionCellTitleIdentifier = @"PhotoCellTitle";

@implementation SBCollectionController

/*  Creates a new collection controller object for the secified fetch request   */
- (SBCollectionController *) initWithFrame:(CGRect) frame fetchRequest:(NSFetchRequest *) request
{
    self = [super initWithFrame: frame];

    if (self) {

        SBAppDelegate *applicationDelegate = (SBAppDelegate *)[[UIApplication sharedApplication] delegate];
        // prepare the fetched results controller, initial the managed object context is nil. Will be added later,
            also set the cache to the hash result of the fetch request
        self.collectionFetchedResultsController = [[NSFetchedResultsController alloc] initWithFetchRequest: request
            managedObjectContext: [applicationDelegate managedObjectContext] sectionNameKeyPath: nil cacheName: nil]
            ;

        // set the controllers delegate
        [self.collectionFetchedResultsController setDelegate: self];

        // create the array for animation of objects
        self.collectionAnimationObjects = [[NSMutableArray alloc] init];
        self.indexPathOfSelectedObject = nil;
        self.collectionFetchRequest = request;

        // the fetched objects should not be returned as faults because we know for sure that other values the the
            ID are needed
        [request setReturnsObjectsAsFaults: NO];

        // create the collection view
        SBCollectionCellViewLayout *layout = [[SBCollectionCellViewLayout alloc] init];
        self.collectionView = [[UICollectionView alloc] initWithFrame: CGRectZero collectionViewLayout: layout];

        // configure the collection view
        [self.collectionView setDelegate: self];
        [self.collectionView setDataSource: self];
        [self.collectionView setBackgroundColor: [UIColor lightGrayColor]];
        [self.collectionView registerClass: [SBElementCollectionViewCell class] forCellWithReuseIdentifier:
            SBCollectionCellIdentifier];

        // register the title view class for a supplementary view object
        [self.collectionView registerClass: [SBCollectionViewCellTitleView class] forSupplementaryViewOfKind:
            SBCollectionViewLayoutSupplementaryKindTitle withReuseIdentifier: SBCollectionCellTitleIdentifier];

        // when the collection results controller is not nil execute the fetch request
        NSError *error = nil;
        if (self.collectionFetchedResultsController != nil && ![self.collectionFetchedResultsController
            performFetch: &error]) {
            NSLog(@"Error when fetching: %@", [error userInfo]);
        }
    }

    return self;
}

/*  Remove observer from notification center    */
- (void) dealloc
{

}

/*  Returns the NSNumber object for the selected object */
- (NSNumber *) identifierForSelectedIndexPath
{
    // return the identifier from the object
    return [(NSManagedObject <SBManagedObjectInterface>*)[self.collectionFetchedResultsController objectAtIndexPath:
```

```
        self.indexPathOfSelectedObject] nextCategoryIdentifier];
}

/*  Retrieve the managed object that is selected by the user    */
- (NSManagedObject *) managedObjectForSelectedIndexPath
{
    // get the object at the indexpath from the fetched results controller
    return [self.collectionFetchedResultsController objectAtIndexPath: self.indexPathOfSelectedObject];
}

/*  When the view will be placed in the superview calculate the size    */
- (void) willMoveToSuperview:(UIView *) newSuperview
{
    // set the collection view frame to the super view frame
    [self.collectionView setFrame: newSuperview.bounds];

    // when not already added to the superview do so
    if ([self.collectionView superview] == nil){
        [self addSubview: self.collectionView];
    }

}

/*  Resets the selected object  */
- (void) reset
{
    // reset the selected row, reset the indexpath and reload the table
    [[self.collectionFetchedResultsController objectAtIndexPath: self.indexPathOfSelectedObject] setIsSelected: NO];
    self.indexPathOfSelectedObject = nil;
    [self.collectionView reloadData];
}

#pragma mark    -
#pragma mark    NSPersistentStorecoordinater change Notifications

/*  When the persistent store coordinator will change its stores, respond to it in this metho by appending the
    fetched results controller   */
- (void) persistentStoreCoordinatorWillChangeStores:(NSNotification *)aNotification
{
    // set the fetched results controller to nil because the store will be removed with objects at the managed
        object context. Will created warnings otherwise
    self.collectionFetchedResultsController = nil;
    [self.collectionView reloadData];
}

/*  when the persistent store coordinator did change its stores respond to it via this method and restore the
    managed object context    */
- (void) persistentStoreCoordinatorDidChangeStores:(NSNotification *)aNotification
{
    // when a new store is added reload the fetched results controller
    if ([[aNotification userInfo] objectForKey: NSAddedPersistentStoresKey] != nil)
    {
        // when the stores is finished with changing its stores create a new fetchedresultscontroller for the
            collectionview
        SBAppDelegate *applicationDelegate = (SBAppDelegate *)[[UIApplication sharedApplication] delegate];
        self.collectionFetchedResultsController = [[NSFetchedResultsController alloc] initWithFetchRequest: self.
            collectionFetchRequest managedObjectContext: [applicationDelegate managedObjectContext]
            sectionNameKeyPath: nil cacheName: nil];

        // set the controllers delegate
        [self.collectionFetchedResultsController setDelegate: self];

        // execute the fetch request
        NSError *error = nil;

        if (![self.collectionFetchedResultsController performFetch: &error]){
            NSLog(@"Error fetching data: %@", [error userInfo]);
        }
    }
}

#pragma mark    -
#pragma mark    NSFetchedResultsController delegate

/*  Called when the fetched results controller will start changing its content  */
- (void) controllerWillChangeContent:(NSFetchedResultsController *)controller
{
    // reset the array that holds the animations to be executed
    [self.collectionAnimationObjects removeAllObjects];
}

/*  Called when the fetched results controller finished changing its content    */
- (void) controllerDidChangeContent:(NSFetchedResultsController *)controller
{
    // perform a batch update for the collection view with the rovided updates
    [self.collectionView performBatchUpdates:
     ^{
```

```
        // loop through all animations
        NSEnumerator *animationEnumerator = [self.collectionAnimationObjects objectEnumerator];
        SBFetchedResultsControllerAnimation *currentAnimation = nil;

        while (currentAnimation = [animationEnumerator nextObject])
        {
            switch ([currentAnimation changeType])
            {
                    // the change type for an insert item, just insert the item at its corresponding indexpath
                case NSFetchedResultsChangeInsert:
                    [self.collectionView insertItemsAtIndexPaths: [NSArray arrayWithObject: [currentAnimation
                        endingIndexPath]]];
                    break;

                    // an object was deleted from the controller, remove it from the indexpath
                case NSFetchedResultsChangeDelete:
                    [self.collectionView deleteItemsAtIndexPaths: [NSArray arrayWithObject: [currentAnimation
                        startingIndexPath]]];
                    break;

                    // move an item from its starting indexpath to its ending indexpath
                case NSFetchedResultsChangeMove:
                    [self.collectionView moveItemAtIndexPath: [currentAnimation startingIndexPath] toIndexPath:
                        [currentAnimation endingIndexPath]];
                    break;

                    // when an item is updated in the background update thet indexpath
                case NSFetchedResultsChangeUpdate:
                    [self.collectionView reloadItemsAtIndexPaths: [NSArray arrayWithObject: [currentAnimation
                        startingIndexPath]]];
                    break;
            }
        }

        // remove all items from the array for memory usage
        [self.collectionAnimationObjects removeAllObjects];
     } completion: nil];
}

/*  When the fetched results controller changes an object in the controller this method gets called */
- (void) controller:(NSFetchedResultsController *)controller didChangeObject:(id)anObject atIndexPath:(NSIndexPath *
    )indexPath forChangeType:(NSFetchedResultsChangeType)type newIndexPath:(NSIndexPath *)newIndexPath
{
    // create the animation to perform the batch update and add it to the array
    SBFetchedResultsControllerAnimation *newAnimation = [[SBFetchedResultsControllerAnimation alloc]
        initWithChangeType: type object: anObject startIndexPath: indexPath endIndexPath: newIndexPath];
    [self.collectionAnimationObjects addObject: newAnimation];
}

#pragma mark    -
#pragma mark    UICollectionView Datasource and Delegate

/*  Returns the amount of items in the specified section    */
- (NSInteger) collectionView:(UICollectionView *)collectionView numberOfItemsInSection:(NSInteger)section
{
    // count is equal to the amount of elements
    id<NSFetchedResultsSectionInfo> sectionInfo = [[self.collectionFetchedResultsController sections] objectAtIndex:
        section];
    NSInteger sectionCounter = [sectionInfo numberOfObjects];

    // when animations should be performed recalculate the number of objects that need to be shown
    NSEnumerator *animationEnumerator = [self.collectionAnimationObjects objectEnumerator];
    SBFetchedResultsControllerAnimation *animation = nil;

    while (animation = [animationEnumerator nextObject]) {

        // an insert animation is not finished yet when in the array, so decrease the counter because the object
            will not exist in the collection view when asked. When a delete change is found the object is actually
            still in the collection view, thats why the counter should be increased
        if ([animation changeType] == NSFetchedResultsChangeInsert)
            sectionCounter--;
        else if ([animation changeType] == NSFetchedResultsChangeDelete)
            sectionCounter++;
    }

    // return the up to date section counter
    return sectionCounter;
}

/*  Returns the amount of sections in the collection view   */
- (NSInteger) numberOfSectionsInCollectionView:(UICollectionView *)collectionView
{
    return [[self.collectionFetchedResultsController sections] count];
}

/*  Returns a cell for the specified Indexpath of the collectionview    */
- (UICollectionViewCell *) collectionView:(UICollectionView *)collectionView cellForItemAtIndexPath:(NSIndexPath *)
```

```
    indexPath
{
    // set the reuse identifier and try to dequeue a cell
    SBElementCollectionViewCell *cell = [collectionView dequeueReusableCellWithReuseIdentifier:
        SBCollectionCellIdentifier forIndexPath: indexPath];

    // download the image for this object
    if (cell.imageView == nil) {

        // create a new cached image operation
        SBURLCachedImageOperation *cacheOperation = [[SBURLCachedImageOperation alloc] initWithManagedObjectID:
            [[self.collectionFetchedResultsController objectAtIndexPath: indexPath] objectID] withOptions: nil];
        [cacheOperation setDelegate: self];

        // start the operation by adding it to the queue
        [[SBOperationManager defaultManager] addOperation: cacheOperation];
    }

    return cell;
}

/*  Fetches the supplementory view for the given index path */
- (UICollectionReusableView *) collectionView:(UICollectionView *)collectionView viewForSupplementaryElementOfKind:
    (NSString *)kind atIndexPath:(NSIndexPath *)indexPath
{
    SBCollectionViewCellTitleView *titleView = [collectionView dequeueReusableSupplementaryViewOfKind:kind
        withReuseIdentifier: SBCollectionCellTitleIdentifier forIndexPath:indexPath];
    NSManagedObject <SBManagedObjectInterface> *currentObject = (NSManagedObject <SBManagedObjectInterface> *)[self.
        collectionFetchedResultsController objectAtIndexPath: indexPath];

    // set the title label description
    titleView.titleLabel.text = [currentObject collectionViewDescription];

    return titleView;
}

/*  When a user selects a cell this method gets called  */
- (void) collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath *)indexPath
{
    // set the current selected object
    self.indexPathOfSelectedObject = indexPath;

    // get the object representing being seleced
    NSNumber *identifier = [(NSManagedObject <SBManagedObjectInterface>*)[self.collectionFetchedResultsController
        objectAtIndexPath: self.indexPathOfSelectedObject] nextCategoryIdentifier];

    // notify the collection delegate that an item was selected
    [self.delegate collectionController: self didSelectItemWithIdentifier: identifier];
}

#pragma mark    -
#pragma mark    SBOperation delegate methods

/*  When the operation failed to load the data replace the image with a default one */
- (void) operation:(SBOperation *) operation didFailWithError:(NSError *) error
{
    // get the managed object
    NSManagedObjectContext *context = [(SBAppDelegate *)[[UIApplication sharedApplication] delegate]
        managedObjectContext];

    // when the image is received find the corresponding indexpath for this object
    SBURLCachedImageOperation *cacheOperation = (SBURLCachedImageOperation *) operation;

    // get the indexpath in the fetchedresultscontrolle for the managed object
    NSIndexPath *indexPath = [self.collectionFetchedResultsController indexPathForObject: [context objectWithID:
        cacheOperation.objectID]];
    SBElementCollectionViewCell *currentCell = (SBElementCollectionViewCell *)[self.collectionView
        cellForItemAtIndexPath: indexPath];

    // prepare cell for default image
    [currentCell prepareForDefaultImage];
}

/*  When the image was loaded for the image this method is called   */
- (void) operation:(SBOperation *)operation didFinishWithResult:(SBAPIResponse *)response
{
    // get the main managed object context for the application
    NSManagedObjectContext *managedObjectContext = [(SBAppDelegate *)[[UIApplication sharedApplication] delegate]
        managedObjectContext];

    // when the image is received find the corresponding index path
    SBURLCachedImageOperation *cacheOperation = (SBURLCachedImageOperation *) operation;
    NSIndexPath *indexPath = [self.collectionFetchedResultsController indexPathForObject: [managedObjectContext
        objectWithID: cacheOperation.objectID]];

    // prepare the cell for the image
    SBElementCollectionViewCell *currentCell = (SBElementCollectionViewCell *)[self.collectionView
```

```
            cellForItemAtIndexPath: indexPath];
        [currentCell prepareForImage: [response imageResource]];
    }

    @end
```

```
//
//  SBCollectionViewCellTitleView.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 13-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface SBCollectionViewCellTitleView : UICollectionReusableView

// the title label for this view
@property (nonatomic, strong) UILabel *titleLabel;

// the used font for drawing the text
@property (nonatomic, strong) UIFont *titleFont;

@end
```

```
//
//  SBCollectionViewCellTitleView.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 13-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBCollectionViewCellTitleView.h"

@implementation SBCollectionViewCellTitleView

/*  Initializes the title view with the provided frame  */
- (SBCollectionViewCellTitleView *) initWithFrame:(CGRect) frame
{
    self = [super initWithFrame: frame];

    if (self) {

        // set the title font
        self.titleFont = [UIFont boldSystemFontOfSize: 13.0f];

        // create the title label with max frame is the bounds of the title supplementory view
        self.titleLabel = [[UILabel alloc] initWithFrame: self.bounds];

        // configure the text label
        self.titleLabel.backgroundColor = [UIColor clearColor];
        self.titleLabel.textAlignment = NSTextAlignmentCenter;
        self.titleLabel.font = self.titleFont;
        self.titleLabel.textColor = [UIColor colorWithWhite: 1.0f alpha: 1.0f];
        self.titleLabel.shadowColor = [UIColor colorWithWhite: 0.0f alpha: 0.3f];
        self.titleLabel.shadowOffset = CGSizeMake(0.0f, 1.0f);

        // set multiple lines to make sure line break will enable
        [self.titleLabel setNumberOfLines: 4];

        // add the view to the title supplementory viwe
        [self addSubview: self.titleLabel];
    }

    return self;
}

/*  When the cell will be reused make sure to call the parent context and reset the text label text */
- (void) prepareForReuse
{
    [super prepareForReuse];

    // reset the text label
    self.titleLabel.text = nil;
}

@end
```

```
//
//  SBCollectionViewLocationCell.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 21-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface SBCollectionViewLocationCell : UICollectionViewCell

// the iboutlet properties of the cell
@property (nonatomic, weak) IBOutlet UIImageView *imageView;
@property (nonatomic, weak) IBOutlet UILabel *titleLabel;
@property (nonatomic, weak) IBOutlet UILabel *footnoteLabel;

@end
```

```
//
//  SBCollectionViewLocationCell.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 21-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBCollectionViewLocationCell.h"

@implementation SBCollectionViewLocationCell
@end
```

```
//
//  SBElementCollectionViewCell.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 10-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBCollectionCellCompatabilityView.h"

@interface SBElementCollectionViewCell : UICollectionViewCell

// the activity indicator to notify image loading
@property (nonatomic, strong) UILabel *elementLabel;
@property (nonatomic, strong) UIActivityIndicatorView *activityIndicator;

// the image shown in the cell
@property (nonatomic, strong) UIImageView *imageView;

// set outlets
@property (nonatomic, strong) SBCollectionCellCompatabilityView *compatabilityView;

// prepares the view for the new image that will be shown
- (void) prepareForImage:(UIImage *) image;
- (void) prepareForDefaultImage;

@end
```

```
//
//  SBElementCollectionViewCell.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 10-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBElementCollectionViewCell.h"
#import "SBViewCompatabilityFactory.h"
#import "SBCollectionCellCompatabilityView.h"

@implementation SBElementCollectionViewCell

- (id) initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {

        // configure the view layer background and border with shadows
        self.backgroundColor = [UIColor colorWithWhite: 0.85f alpha: 1.0f];
        self.layer.borderColor = [UIColor whiteColor].CGColor;
        self.layer.borderWidth = 3.0f;
        self.layer.shadowColor = [UIColor blackColor].CGColor;
        self.layer.shadowRadius = 3.0f;
        self.layer.shadowOffset = CGSizeMake(0.0f, 2.0f);
        self.layer.shadowOpacity = 0.5f;

        // the activity indicator for the image loading
        self.activityIndicator = [[UIActivityIndicatorView alloc] initWithActivityIndicatorStyle:
            UIActivityIndicatorViewStyleWhite];

        // set the activity frame the center of the frame of this object
        [self.activityIndicator setCenter: CGPointMake(self.bounds.size.width / 2, self.bounds.size.height / 2)];
        [self.contentView addSubview: self.activityIndicator];

        // start the animation
        [self.activityIndicator startAnimating];

        // set rastorization configurations for performance
        self.layer.rasterizationScale = [[UIScreen mainScreen] scale];
        self.layer.shouldRasterize = YES;
    }
    return self;
}

/*  Listen to when the object gets selected, append the border color when this occures  */
- (void) setSelected:(BOOL) selected
{
    // handle the setting of the selected boolean by super
    [super setSelected: selected];

    // change the border color to let know that the item is selected or not
    if (selected) {

        // when selected the border is dark gray
        self.layer.borderColor = [UIColor darkGrayColor].CGColor;
    } else {

        // otherwise the border is plain white
        self.layer.borderColor = [UIColor whiteColor].CGColor;
    }
}

/*  Prepare for a new image that is provided */
- (void) prepareForImage:(UIImage *)image
{
    // when the image view is already existing remove it from its superview
    if (self.imageView != nil) {
        [self.imageView removeFromSuperview];
        self.imageView = nil;
    }

    // when the activity indicator is still animating release it
    if (self.activityIndicator != nil) {

        // hide and stop the loading animation
        [self.activityIndicator stopAnimating];
        [self.activityIndicator removeFromSuperview];
        self.activityIndicator = nil;
    }

    // create the image view
    self.imageView = [[UIImageView alloc] initWithImage: image];

    // set the max frame of the view
    [self.imageView setFrame: self.bounds];
```

```
    [self.imageView setAlpha: 0.0];
    [self.contentView addSubview: self.imageView];
    [self.imageView setAlpha: 1.0];
}

/*  Sets the default image visible  */
- (void) prepareForDefaultImage
{
    // when the image view is already existing remove it from its superview
    if (self.imageView != nil) {
        [self.imageView removeFromSuperview];
        self.imageView = nil;
    }

    // when the activity indicator is still animating release it
    if (self.activityIndicator != nil) {
        [self.activityIndicator stopAnimating];
        [self.activityIndicator removeFromSuperview];
        self.activityIndicator = nil;
    }

    // create the image view
    self.imageView = [[UIImageView alloc] initWithImage: [UIImage imageNamed: @"defectNotFound"]];

    // set the max frame of the view
    [self.imageView setFrame: self.bounds];
    [self.imageView setAlpha: 0.0];
    [self.contentView addSubview: self.imageView];

    // animate the view on screen
    [UIView animateWithDuration: 0.1 animations:^{

        // insert the image view to the cell view
        [self.imageView setAlpha: 1.0];
    }];
}

/*  When the cell will be reused reset the text in the textlabel    */
- (void) prepareForReuse
{
    // when reusing reset the text in the label and call super method
    [super prepareForReuse];
    self.elementLabel.text = @"";
}

@end
```

```
//
//  SBErrorView.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 22-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

typedef enum {
    SBErrorImageTypeWarning     =   0,  // shows a yellow warning image
    SBErrorImageTypeSuccess     =   1,  // shows a green success image
    SBErrorImageTypeFailed      =   2,   // shows an red error sign image
    SBErrorImageTypeLoading     =   3,   // shws an activity indicator
    SBErrorImageTypeNone        =   4
} SBErrorImageType;

@class SBErrorView, SBErrorCompatabilityView;

// the function signature for the callback of touches
typedef void (^touchCallback) (SBErrorView *,UITapGestureRecognizer *);

@interface SBErrorView : UIView
{
    // callback method for a touch
    touchCallback errorViewTouchCallback;
}

// the error message
@property (nonatomic, strong) NSString *errorMessage;

// image type and wether it should respond to touches
@property (nonatomic) SBErrorImageType imageType;
@property (nonatomic) BOOL shouldRespondToTouches;
@property (nonatomic, strong) UITapGestureRecognizer *gestureRecognizer;
@property (nonatomic, strong) SBErrorCompatabilityView *compatabilityView;

// initializer method for this error view, also declare method that handles touches
- (SBErrorView *) initWithFrame:(CGRect) frame errorMessage:(NSString *) message withImageType:(SBErrorImageType)
    type;
- (void) shouldHandleTouches:(BOOL) yesNo withCallback:(touchCallback) callback;

// when a touch is received this method is called
- (void) tapGestureReceived:(UITapGestureRecognizer *) sender;

// method returns the image view or activity view for the current message rectangle and image type
- (UIView *) returnImageViewForMessageRect:(CGRect) messageRect;

@end
```

```
//
//  SBErrorView.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 22-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBErrorView.h"
#import "SBViewCompatabilityFactory.h"
#import "SBErrorCompatabilityView.h"

@implementation SBErrorView

/*  Custom initializer with an error message and image type */
- (SBErrorView *) initWithFrame:(CGRect)frame errorMessage:(NSString *)message withImageType:(SBErrorImageType)type
{
    self = [super initWithFrame:frame];

    if (self)
    {
        // Initialization code
        self.errorMessage = message;
        self.imageType = type;

        // get the compatability draw controller
        self.compatabilityView = [SBViewCompatabilityFactory loadCompatableErrorController];

        // background color is clear
        [self setBackgroundColor: [UIColor clearColor]];
    }

    return self;
}

/*  Returns and calculate the frame for the image view or activity indicator view   */
- (UIView *) returnImageViewForMessageRect:(CGRect) messageRect
{
    // create the view to return
    UIView *createdView = nil;

    // calculate the view rect for this image
    CGFloat x = (self.frame.size.width - 32) / 2;
    CGFloat halfViewHeight = (self.frame.size.height / 2) - (messageRect.size.height / 2);
    CGFloat y = (halfViewHeight - 32);
    CGRect imageRect = CGRectMake(x, y, 32, 32);

    // switch through the options to create the selected view type
    switch (self.imageType)
    {
        // no image should be shown
        case SBErrorImageTypeNone:
            break;

        // a green image should be shown
        case SBErrorImageTypeSuccess:
        {
            // get the image for the provided name
            UIImageView *imageView = [[UIImageView alloc] initWithFrame: imageRect];
            [imageView setImage: [UIImage imageNamed: @"Success.png"]];

            // copy the view pointer
            createdView = imageView;
            break;

        }

        // a loading indicator that is spinning should be shown
        case SBErrorImageTypeLoading:
        {
            // create the activity indicator, set the frame and start the animation
            UIActivityIndicatorView *activityIndicator = [[UIActivityIndicatorView alloc]
                initWithActivityIndicatorStyle: UIActivityIndicatorViewStyleGray];
            [activityIndicator setFrame: imageRect];
            [activityIndicator setCenter: CGPointMake(CGRectGetMidX(imageRect), CGRectGetMidY(imageRect))];
            [activityIndicator startAnimating];

            // set the view to return
            createdView = activityIndicator;
            break;
        }

        // a red image icon should be shown
        case SBErrorImageTypeFailed:
        {
            break;
        }
```

```
        case SBErrorImageTypeWarning:
        {
            // get the image for the provided name
            UIImageView *imageView = [[UIImageView alloc] initWithFrame: imageRect];
            [imageView setImage: [UIImage imageNamed: @"Warning.png"]];

            // copy the view pointer
            createdView = imageView;
            break;
        }

    }

    return createdView;
}

/*  Method sets or unsets the touch handling and clears or sets the block to execute    */
- (void) shouldHandleTouches:(BOOL) yesNo withCallback:(touchCallback) callback
{
    // when enabled set the block
    if (yesNo)
    {
        // set the callback block
        errorViewTouchCallback = callback;

        // initialize the gesture recognizer when needed
        if (self.gestureRecognizer == nil) {
            self.gestureRecognizer = [[UITapGestureRecognizer alloc] initWithTarget: self action: @selector
                (tapGestureReceived:)];

            // add the gesture to the view
            [self addGestureRecognizer: self.gestureRecognizer];
        }
    }
    else
    {
        errorViewTouchCallback = nil;
        self.gestureRecognizer = nil;
    }

    // toggle the boolean
    self.shouldRespondToTouches = yesNo;
}

/*  Method implememtation when a user taps this view    */
- (void) tapGestureReceived:(UITapGestureRecognizer *)sender
{
    // as the gesture and object to the callback block
    if (errorViewTouchCallback != nil){
        errorViewTouchCallback(self, sender);
    }
}


// Only override drawRect: if you perform custom drawing.
// An empty implementation adversely affects performance during animation.
- (void) drawRect:(CGRect) rect
{
    // redraw the view using the compatability view, avoiding deprecated and unavailable methods
    [self.compatabilityView drawRect: rect forErrorViewController: self];
}


@end
```

```
//
//  SBHomeTableViewCell.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 16-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface SBHomeTableViewCell : UITableViewCell

// declare the iboutlets of the label and image
@property (nonatomic, weak) IBOutlet UIImageView *actionImageView;
@property (nonatomic, weak) IBOutlet UILabel *actionTextLabel;

// the italic text under the title of the cell
@property (nonatomic, strong) NSString *cellDescription;

// the activity indicator and whether it is hidden
@property (nonatomic) BOOL shouldIndicateActivity;

@end
```

```
//
//  SBHomeTableViewCell.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 16-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBHomeTableViewCell.h"

@interface SBHomeTableViewCell()

// the activity indicator used for identifying activity
@property (nonatomic, strong) UIActivityIndicatorView *activityIndicator;

// the description cell label
@property (nonatomic, strong) UILabel *descriptionLabel;

@end

@implementation SBHomeTableViewCell

/*  Initializes the cell with the given style and reuseidentifier   */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    self = [super initWithCoder: aDecoder];

    if (self) {

        self.shouldIndicateActivity = NO;

        // create the activity indicator
        self.activityIndicator = [[UIActivityIndicatorView alloc] initWithActivityIndicatorStyle:
            UIActivityIndicatorViewStyleGray];
        [self.activityIndicator setFrame: CGRectMake(270, (self.frame.size.height - self.activityIndicator.frame.
            size.height) / 2, self.activityIndicator.frame.size.width, self.activityIndicator.frame.size.height)];
        [self.activityIndicator setHidesWhenStopped: YES];

        // add to the superview
        [self addSubview: self.activityIndicator];
    }

    return self;
}

/*  Gets called when the cell needs to be displayed ons creen   */
- (void) setNeedsDisplay
{
    // call super implementation
    [super setNeedsDisplay];
}

- (void) layoutSubviews
{
    [super layoutSubviews];

    // when a cell description is available append the cell frames
    if (self.cellDescription != nil) {

        // set the frame of the title and description labales
        [self.actionTextLabel setFrame: CGRectMake(self.actionTextLabel.frame.origin.x, 10, self.actionTextLabel.
            frame.size.width, self.actionTextLabel.frame.size.height)];
        [self.descriptionLabel setFrame: CGRectMake(self.actionTextLabel.frame.origin.x, (self.actionTextLabel.frame
            .size.height + 10), self.actionTextLabel.frame.size.width, self.actionTextLabel.frame.size.height)];
    }
}

/*  When the activity indiactor visible boolean is set respond to changes   */
- (void) setShouldIndicateActivity:(BOOL) shouldIndicateActivity
{
    // set the new boolean value
    _shouldIndicateActivity = shouldIndicateActivity;

    // stop or start the animation depending on the new bool value
    if (shouldIndicateActivity) {
        [self.activityIndicator startAnimating];
    } else {
        [self.activityIndicator stopAnimating];
    }
}

/*  When the cell will be reused make sure the image view and text field are empty  */
- (void) prepareForReuse
{
    // call super implementation
    [super prepareForReuse];
```

```
    // set the old frame of the title label
    [self.actionTextLabel setFrame: CGRectMake(self.actionTextLabel.frame.origin.x, 17, self.actionTextLabel.frame.
        size.width, self.actionTextLabel.frame.size.height)];

    // remove the description label from its superview
    if ([self.descriptionLabel superview] != nil) {
        [self.descriptionLabel removeFromSuperview];
    }

    // reset image and label
    [self.actionImageView setImage: nil];
    [self.actionTextLabel setText: nil];

    // reset labels
    self.cellDescription = nil;
    self.shouldIndicateActivity = NO;
    self.descriptionLabel = nil;
}

/*  When the cell description is set the frames should be updated   */
- (void) setCellDescription:(NSString *) cellDescription
{
    // set the new cell description
    _cellDescription = cellDescription;

    // when the new cell description is not nil create the label and calculate frames
    if (_cellDescription != nil) {

        // move the cell title frame up by 10 points
        [self.actionTextLabel setFrame: CGRectMake(self.actionTextLabel.frame.origin.x, 10, self.actionTextLabel.
            frame.size.width, self.actionTextLabel.frame.size.height)];

        // when the description label is not created create it
        if (self.descriptionLabel == nil) {

            // create and configure the label
            self.descriptionLabel = [[UILabel alloc] initWithFrame: CGRectZero];
            [self.descriptionLabel setFont: [UIFont italicSystemFontOfSize: 11.0f]];
            [self.descriptionLabel setText: cellDescription];

            // add to the view
            [self addSubview: self.descriptionLabel];
        }
    }

    // redraw content
    [self setNeedsDisplay];
}



@end
```

```
//
//  SBKeyboardCustomToolbar.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 13-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

@class SBKeyboardToolbarCompatabilityView;

@interface SBKeyboardCustomToolbar : UIToolbar

// this keyboard holds the UILabel for indicating the current character count
@property (nonatomic, strong) UILabel *characterCountLabel;
@property (nonatomic, strong) UIBarButtonItem *doneButton;
@property (nonatomic, strong) SBKeyboardToolbarCompatabilityView *compatabilityView;

// should redraw
@property (nonatomic) BOOL shouldRedraw;

// initializer method
- (SBKeyboardCustomToolbar *) initWithButtonTarget:(id) target withSelector:(SEL) objectSelector frame:(CGRect) rect
    ;
- (void) redraw;

@end
```

```
//
//  SBKeyboardCustomToolbar.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 13-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBKeyboardCustomToolbar.h"
#import "SBViewCompatabilityFactory.h"
#import "SBKeyboardToolbarCompatabilityView.h"

@implementation SBKeyboardCustomToolbar

/*  Create the toolbar which will be placed on top of a keyboard to hold different options  */
- (SBKeyboardCustomToolbar *) initWithButtonTarget:(id)target withSelector:(SEL)objectSelector frame:(CGRect)rect
{
    self = [super initWithFrame:rect];

    if (self) {
        // set needs redraw
        self.shouldRedraw = NO;

        // load the compatability controller
        self.compatabilityView = [SBViewCompatabilityFactory loadCompatableKeyboardToolbarController];

        // create the done button with the appropriate target and selector
        self.doneButton = [[UIBarButtonItem alloc] initWithTitle: NSLocalizedString(@"button.title.next", nil)
            style: UIBarButtonItemStyleBordered target: target action: objectSelector];

        // create the label with the corresponding frame
        self.characterCountLabel = [[UILabel alloc] initWithFrame: CGRectMake(0, 0, 100, rect.size.height)];
        [self.characterCountLabel setText: @"0 / 256"];
        [self.characterCountLabel setBackgroundColor: [UIColor clearColor]];

        // set italic text style for the label
        [self.characterCountLabel setFont: [UIFont italicSystemFontOfSize: [UIFont systemFontSize]]];

        // create a white space in between the bar items
        UIBarButtonItem *whiteSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
            UIBarButtonSystemItemFlexibleSpace target: nil action: nil];
        UIBarButtonItem *labelContainer = [[UIBarButtonItem alloc] initWithCustomView: self.characterCountLabel];

        // set the items of the toolbar
        [self setItems: [NSArray arrayWithObjects: labelContainer, whiteSpace, self.doneButton, nil]];

        // config using the compatability controller
        [self.compatabilityView initializeInterfaceOfViewController: self];
    }

    return self;
}

/*  Redraws the text label  */
- (void) redraw
{
    // when needs redraw use the compatability view to redraw
    if (self.shouldRedraw) {

        [self.compatabilityView initializeInterfaceOfViewController: self];
        self.shouldRedraw = NO;
    }
}

@end
```

```
//
//  SBKeyboardToolbar.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 19-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBKeyboardToolbarLayout.h"
#import "SBKeyboardToolbarDelegate.h"

@interface SBKeyboardToolbar : UIToolbar

// the current layout object used for the toolbar
@property (nonatomic, strong) SBKeyboardToolbarLayout *currentLayout;

// the delegate object for the custom toolbar methods
@property (nonatomic, strong) id<SBKeyboardToolbarDelegate> delegateObject;

// method that redesigns the toolbar layout with the new layout object
- (void) redrawWithLayout:(SBKeyboardToolbarLayout *) newLayout;

@end
```

```
//
//  SBKeyboardToolbar.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 19-09-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBKeyboardToolbar.h"
#import "SBViewCompatabilityFactory.h"

@interface SBKeyboardToolbar()

// method gets called when the left bar button was pressed
- (IBAction) tappedLeftBarButtonItem:(UIBarButtonItem *) sender;

// method gets called when the right bar button item was pressed
- (IBAction) tappedRightBarButtonItem:(UIBarButtonItem *) sender;

// redraws and processes the layout
- (void) processLayout;

@end

@implementation SBKeyboardToolbar

/*  This method processes the toolbar items to the new layout   */
- (void) processLayout
{
    // create a flexible width between toolbar views
    UIBarButtonItem *flexibleSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
        UIBarButtonSystemItemFlexibleSpace target: nil action: nil];
    NSMutableArray *toolbarItems = [NSMutableArray array];

    // add objects for the first item when needed
    if (self.currentLayout.leftBarButtonItem != nil) {
        [toolbarItems addObject: self.currentLayout.leftBarButtonItem];
    }

    // add string items
    if (self.currentLayout.title != nil) {

        // when a previous item has been added first add the flexible space
        [toolbarItems addObject: flexibleSpace];

        // create the font used by the label
        UIFont *textFont = [UIFont boldSystemFontOfSize: 16.0f];

        // create the label fr the toolbar
        CGSize frame = [SBViewCompatabilityFactory boundingRectForString: self.currentLayout.title withSize:
            CGSizeMake(150, 44) options: NSStringDrawingUsesFontLeading attributes: [NSDictionary
            dictionaryWithObject: textFont forKey: NSFontAttributeName] context: nil];

        // create and confgure the text label for the toolbar
        UILabel *titleLabel = [[UILabel alloc] initWithFrame: CGRectMake(0, 0, frame.width, frame.height)];
        [titleLabel setText: self.currentLayout.title];
        [titleLabel setFont: textFont];
        [titleLabel setTextAlignment: NSTextAlignmentCenter];
        [titleLabel setBackgroundColor: [UIColor clearColor]];

        UIBarButtonItem *titleView = [[UIBarButtonItem alloc] initWithCustomView: titleLabel];
        [toolbarItems addObject: titleView];
    }

    // add the right bar button
    if (self.currentLayout.rightBarButtonItem != nil) {
        [toolbarItems addObject: flexibleSpace];
        [toolbarItems addObject: self.currentLayout.rightBarButtonItem];
    }

    // set the new toolbar items
    [self setItems: toolbarItems animated: NO];
    [self sizeToFit];
}

/*  Prepare the toolbar for the new layout  */
- (void) redrawWithLayout:(SBKeyboardToolbarLayout *) newLayout
{
    // when the left bar button is set process the listening object
    if ([newLayout leftBarButtonItem] != nil) {

        // set target and selector
        [[newLayout leftBarButtonItem] setTarget: self];
        [[newLayout leftBarButtonItem] setAction: @selector(tappedLeftBarButtonItem:)];
    }

    // also when the right bar button is pressed process the listening object
```

```
    if ([newLayout rightBarButtonItem] != nil) {

        // set target and selector
        [[newLayout rightBarButtonItem] setTarget: self];
        [[newLayout rightBarButtonItem] setAction: @selector(tappedRightBarButtonItem:)];
    }

    // set the new layout object and redraw
    [self setCurrentLayout: newLayout];
    [self processLayout];
}

/*  When the left bar button was tapped notify the delegate */
- (IBAction) tappedLeftBarButtonItem:(UIBarButtonItem *)sender
{
    // when the delegate object is able to respond call method
    if ([self.delegateObject respondsToSelector: @selector(keyboardToolbarLeftBarButtonTapped:)]) {
        [self.delegateObject keyboardToolbarLeftBarButtonTapped: self];
    }
}

/*  When the right bar button was tapped notify the delegate    */
- (IBAction) tappedRightBarButtonItem:(UIBarButtonItem *)sender
{
    // when the delegate object is able to respond call method
    if ([self.delegateObject respondsToSelector: @selector(keyboardToolbarRightBarButtonTapped:)]) {
        [self.delegateObject keyboardToolbarRightBarButtonTapped: self];
    }
}

@end
```

```
//
//  SBNavigationBar_iOS_6_0.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 28-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBNavigationBar.h"

@interface SBNavigationBar_iOS_6_0 : SBNavigationBar
@end
```

```
//
//  SBNavigationBar.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 16-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBNavigationBar_iOS_6_0.h"
#import "SBNavigationBarDetailView.h"
#import "SBNavigationCompatability.h"
#import "SBViewCompatabilityFactory.h"
#import <Availability.h>

@implementation SBNavigationBar_iOS_6_0

/*  Initializes the navigation bar with provided frame  */
- (id) initWithFrame:(CGRect)frame
{
    self = [super initWithFrame: frame];

    if (self) {

        // initialize the bar colors etc via compatabiliyu
        // set the navigation bar title attributes, the font should be smaller than normal, the text color should be
            set to black in case of the white navigation bar background. And finally the text shadow should be
            removed
        [self setTitleTextAttributes: @{UITextAttributeTextColor: [UIColor blackColor],
            UITextAttributeTextShadowOffset: [NSValue valueWithCGSize: CGSizeZero], UITextAttributeFont: [UIFont
            boldSystemFontOfSize: 18.0]}];

        // set the background color of the navigation bar
        [self setBackgroundColor: [UIColor whiteColor]];
        [self setTintColor: [UIColor whiteColor]];

        // initialize current count with 0
        self.currentViewControllerCount = 0;
        self.shouldAcceptNextDetailView = YES;
    }

    return self;
}

/*  Layout all subviews needed for the navigation bar   */
- (void) layoutSubviews
{
    // call super implementation to layout the default objects
    [super layoutSubviews];

    // calculate adjustemtn for vertical coordinates
    CGFloat verticalAdjustment = (0 - (self.frame.size.height - 44));

    // set the title vertical adjustment according to the current height of the frame, the title will be set to the
        correct position this way. "44" is the default navigation bar height for holding the title
    [self setTitleVerticalPositionAdjustment: verticalAdjustment forBarMetrics: UIBarMetricsDefault];
}

/*  Implement method to make sure a horizontal line is not drawn by the super class */
- (void) drawRect:(CGRect)rect
{
    // no implementation, only prevents super drawing
}

@end
```

```
//
//  SBNavigationBar.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 16-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBNavigationBar.h"

@interface SBNavigationBar_iOS_7_0 : SBNavigationBar
@end
```

```
//
//  SBNavigationBar.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 16-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBNavigationBar_iOS_7_0.h"
#import "SBNavigationBarDetailView.h"
#import "SBNavigationCompatability.h"
#import "SBViewCompatabilityFactory.h"
#import <Availability.h>

@implementation SBNavigationBar_iOS_7_0

/*  Initializes the navigation bar with provided frame  */
- (id) initWithFrame:(CGRect)frame
{
    self = [super initWithFrame: frame];

    if (self) {

        // initialize the bar colors etc for correct iOS 7 compatability
        [self setBarTintColor: [UIColor whiteColor]];
        [self setTintColor: [UIColor blackColor]];
        [self setBackgroundColor: [UIColor whiteColor]];
        [self setTranslucent: NO];

        // initialize current count with 0
        self.currentViewControllerCount = 0;
        self.shouldAcceptNextDetailView = YES;
    }

    return self;
}

/*  Layout all subviews needed for the navigation bar   */
- (void) layoutSubviews
{
    // call super implementation to layout the default objects
    [super layoutSubviews];

    // calculate adjustemtn for vertical coordinates
    CGFloat verticalAdjustment = (0 - (self.frame.size.height - 44));

    // adjust the back bar button frame
    for (UIView *subview in self.subviews) {

        // check for the back indicator button
        if ([subview isKindOfClass: NSClassFromString(@"_UINavigationBarBackground")]) {

            // remove all subviews from this view, normally only an imageview is added to this view, this image view
                represents the line at the bottom of the navigation bar. In our implementation we do not want this
                line
            if ([[subview subviews] count] > 0) {
                [[[subview subviews] objectAtIndex: 0] removeFromSuperview];
            }
        }
    }

    // set the title vertical adjustment according to the current height of the frame, the title will be set to the
        correct position this way. "44" is the default navigation bar height for holding the title
    [self setTitleVerticalPositionAdjustment: verticalAdjustment forBarMetrics: UIBarMetricsDefault];
}

@end
```

```
//
//  SBNAvigationBar.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 28-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

@class SBNavigationBarDetailView;

// animation identifiers
static NSString *SBNavigationAnimationOldMoveOut    =   @"CABasicMoveOutOldAnimation";
static NSString *SBNavigationAnimationNewMoveIn     =   @"CABasicMoveInNewAnimation";

@interface SBNavigationBar : UINavigationBar

// property for the current and next detail view that will be presented
@property (nonatomic, strong) SBNavigationBarDetailView *currentDetailView;
@property (nonatomic, strong) SBNavigationBarDetailView *nextDetailView;

// whether the detail view conversion should be animated
@property (nonatomic) BOOL shouldAcceptNextDetailView;

// sets the new detail view
- (void) setDetailView:(SBNavigationBarDetailView *) newView;

@end

@interface SBNavigationBar()

// returns an animation object that will be used to animate the old detail view out
- (CAAnimation *) animationForOldDetailViewWithPush:(BOOL) pushOrPop;

// returns an animation for the new detail view moving on screen
- (CAAnimation *) animationForNewDetailViewWithPush:(BOOL) pushOrPop;

// the counter that holds the current amount of view controllers in the navigation stack, is used to know when
    pushed or popped
@property (nonatomic) NSUInteger currentViewControllerCount;

@end
```

```
//
//  SBNAvigationBar.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 28-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBNavigationBar.h"
#import "SBNavigationBarDetailView.h"

@implementation SBNavigationBar

/*  When the detail view gets set invalidate the layout of the view */
- (void) setDetailView:(SBNavigationBarDetailView *) detailView
{
    // when the next detail view should not be accepted return immediatly
    if (!self.shouldAcceptNextDetailView) {
        self.shouldAcceptNextDetailView = YES;
        return;
    }

    // get the navigation bar delegate and whether a pop or piush action should be executed
    UINavigationController *navigationDelegate = (UINavigationController *)self.delegate;
    BOOL shouldPush = YES;

    // when higher than the previous value a push action should be shown
    if (navigationDelegate.viewControllers.count < self.currentViewControllerCount) {
        shouldPush = NO;
    }

    // set the new counter
    self.currentViewControllerCount = navigationDelegate.viewControllers.count;

    // when a detail view is already added remove it from the view
    if ([self.currentDetailView superview] != nil) {

        // set the next detail view
        self.nextDetailView = detailView;
        [self addSubview: self.nextDetailView];

        // start the animation for the new and old detail content view
        [[self.currentDetailView layer] addAnimation: [self animationForOldDetailViewWithPush: shouldPush] forKey:
            SBNavigationAnimationOldMoveOut];
        [[self.nextDetailView layer] addAnimation: [self animationForNewDetailViewWithPush: shouldPush] forKey:
            SBNavigationAnimationNewMoveIn];

    } else {

        // set the current detail view
        self.currentDetailView = detailView;
        [self addSubview: self.currentDetailView];
    }

    // invalidate layout
    [self sizeToFit];
    [self setNeedsLayout];
}

/*  Returns the actual size that will be used for the navigation bar, this method adds the height of the detailed
    view to the navigation bar size to make sure all content is shown */
- (CGSize) sizeThatFits:(CGSize) size
{
    // get the super calculated size, by default this is in most cases {320,44}
    CGSize superSize = [super sizeThatFits: size];

    // calculate the size of the detail view that will be shown in the view
    CGSize detailSize = [self.currentDetailView sizeThatFits: CGSizeZero];

    // return the appended size by calculating the detailed view size in the navigation bar height
    return CGSizeMake(superSize.width, superSize.height + detailSize.height);
}

#pragma mark    -
#pragma mark    CAAnimation methods

/*  Creates and returns the animation that will be used to move the old detail view out */
- (CAAnimation *) animationForOldDetailViewWithPush:(BOOL) pushOrPop
{
    // create the animation group which holds the animations for the old view moving out
    CAAnimationGroup *oldAnimationGroup = [CAAnimationGroup animation];

    // create the animation for bounds and opacity
    CABasicAnimation *moveOutAnimation = [CABasicAnimation animationWithKeyPath: @"bounds"];
    CABasicAnimation *opacityAnimation = [CABasicAnimation animationWithKeyPath: @"opacity"];

    // the default origin x value
```

```
    CGFloat originX = (self.currentDetailView.bounds.origin.x + self.bounds.size.width);

    // when no push action should be executed recalculate the x origin
    if (!pushOrPop) {
        originX = (self.currentDetailView.bounds.origin.x - self.bounds.size.width);
    }

    // set the animation start value and the value to which the animation should move the object
    [moveOutAnimation setFromValue: [NSValue valueWithCGRect: self.currentDetailView.bounds]];
    [moveOutAnimation setToValue: [NSValue valueWithCGRect: CGRectMake(originX, self.currentDetailView.bounds.origin
        .y, self.currentDetailView.bounds.size.width, self.currentDetailView.bounds.size.height)]];

    // set the from and to value of the opacity animation
    [opacityAnimation setFromValue: [NSNumber numberWithFloat: 1.0f]];
    [opacityAnimation setToValue: [NSNumber numberWithFloat: 0.0f]];

    // add the animations to the group and return the group animation
    [oldAnimationGroup setAnimations: [NSArray arrayWithObjects: moveOutAnimation, opacityAnimation, nil]];
    [oldAnimationGroup setDelegate: self];
    [oldAnimationGroup setRemovedOnCompletion: NO];

    // set fill mode to make sure the changes to the view keep until removed
    [oldAnimationGroup setFillMode: kCAFillModeBoth];
    [oldAnimationGroup setDuration: 0.35f];

    // the old view will also move out of the screen very quickly using the specified bezier path, the deffault one
        that is
    [oldAnimationGroup setTimingFunction: [CAMediaTimingFunction functionWithName: kCAMediaTimingFunctionDefault]];

    // return the group
    return oldAnimationGroup;
}

/*  Get the animation which moves the new content in    */
- (CAAnimation *) animationForNewDetailViewWithPush:(BOOL) pushOrPop
{
    // create the animation group which holds the animations for the old view moving out
    CAAnimationGroup *oldAnimationGroup = [CAAnimationGroup animation];

    // create the animation for bounds and opacity
    CABasicAnimation *moveOutAnimation = [CABasicAnimation animationWithKeyPath: @"bounds"];
    CABasicAnimation *opacityAnimation = [CABasicAnimation animationWithKeyPath: @"opacity"];

    // the default origin x value
    CGFloat originX = (self.currentDetailView.bounds.origin.x - self.bounds.size.width);

    // when no push action should be executed recalculate the x origin
    if (!pushOrPop) {
        originX = (self.currentDetailView.bounds.origin.x + self.bounds.size.width);
    }

    // set the animation start value and the value to which the animation should move the object
    [moveOutAnimation setToValue: [NSValue valueWithCGRect: self.currentDetailView.bounds]];
    [moveOutAnimation setFromValue: [NSValue valueWithCGRect: CGRectMake(originX, self.currentDetailView.bounds.
        origin.y, self.currentDetailView.bounds.size.width, self.currentDetailView.bounds.size.height)]];

    // set the from and to value of the opacity animation
    [opacityAnimation setFromValue: [NSNumber numberWithFloat: 0.0f]];
    [opacityAnimation setToValue: [NSNumber numberWithFloat: 1.0f]];

    // add the animations to the group and set its delegate
    [oldAnimationGroup setAnimations: [NSArray arrayWithObjects: moveOutAnimation, opacityAnimation, nil]];
    [oldAnimationGroup setDelegate: self];
    [oldAnimationGroup setRemovedOnCompletion: NO];

    // set fill mode to make sure the changes to the view keep until removed
    [oldAnimationGroup setFillMode: kCAFillModeBoth];
    [oldAnimationGroup setDuration: 0.35f];

    // whether pop or push the new detail view will always have a bezier path for animation that is default for the
        navigation bar in the current context
    [oldAnimationGroup setTimingFunction: [CAMediaTimingFunction functionWithName: kCAMediaTimingFunctionDefault]];

    // return the group
    return oldAnimationGroup;
}

/*  When an animation has ended */
- (void) animationDidStop:(CAAnimation *) animation finished:(BOOL) finished
{
    // when the animation finished is equal to the old view moving out remove this view
    if ([animation isEqual: [[self.currentDetailView layer] animationForKey: SBNavigationAnimationOldMoveOut]]) {

        // remove all animations and remove the view from its superview
        [self.currentDetailView removeFromSuperview];

        // now apply the next detail view to the current one
```

```
            self.currentDetailView = self.nextDetailView;
        }
    }

    @end
```

```
//
//  SBNavigationControllerExtendedViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 14-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface SBNavigationControllerExtended : UINavigationController <UINavigationBarDelegate>

@end
```

```
//
//  SBNavigationControllerExtendedViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 14-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBNavigationControllerExtended.h"
#import "SBPhoneDescriptionViewController.h"

@interface UINavigationController ()

// allows for parent calling of the navigation bar popping
- (BOOL) navigationBar:(UINavigationBar *)navigationBar shouldPopItem:(UINavigationItem *)item;

@end

@implementation SBNavigationControllerExtended

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

/*  When a new item is pushed validate wether this is possible  */
- (BOOL) navigationBar:(UINavigationBar *)navigationBar shouldPopItem:(UINavigationItem *)item
{
    // set the default view controller
    UIViewController *currentViewController = nil;

    // check whether the navigation item is the one of the description class
    for (currentViewController in [self viewControllers]) {
        // when the navigation items are equal break and use the current view controller
        if ([[currentViewController navigationItem] isEqual: item]) {
            break;
        }
    }

    // when the description class is found start the keyboard dismissal
    if ([currentViewController isKindOfClass: [SBPhoneDescriptionViewController class]]) {
        SBPhoneDescriptionViewController *controller = (SBPhoneDescriptionViewController *) currentViewController;
        [controller popViewController];
        return NO;
    }

    // when the description view controller is not found allow popping
    return [super navigationBar: navigationBar shouldPopItem: item];
}

@end
```

```
//
//  SBReservationOverviewCell.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 26-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

@class Reservation;

@interface SBReservationOverviewCell : UITableViewCell

// create the outlets for the overview cell
@property (nonatomic, weak) IBOutlet UIImageView *reservationImage;
@property (nonatomic, weak) IBOutlet UILabel *reservationNameLabel;
@property (nonatomic, weak) IBOutlet UILabel *reservationDateLabel;
@property (nonatomic, weak) IBOutlet UILabel *reservationTimeLabel;
@property (nonatomic, weak) IBOutlet UILabel *reservationStatusLabel;

// the reservation object corresponding to this cell
@property (nonatomic, strong) Reservation *reservation;

// the progress view used to show the status of uploading
@property (nonatomic, strong) UIProgressView *progressView;

@end
```

```
//
//  SBReservationOverviewCell.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 26-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBReservationOverviewCell.h"
#import "Reservation.h"
#import "Defect.h"
#import "Proceeding.h"

@implementation SBReservationOverviewCell

/*  When the reservation object is set start observing this objects "currentStatus" */
- (void) setValue:(id)value forKey:(NSString *)key
{
    // only respond to the reservation being set
    if ([key isEqualToString: @"reservation"]) {

        // add this object to be observed, when the status chengas we can updated the corresponding cell
        [value addObserver: self forKeyPath: @"currentStatus" options: NSKeyValueObservingOptionNew context: NULL];
    }

    // call super setvalue for key
    [super setValue: value forKey: key];
}

/*  Reset all values when being reused  */
- (void) prepareForReuse
{
    // call super object to prepare for reuse
    [super prepareForReuse];

    // reset all labels
    [self.reservationNameLabel setText: @""];
    [self.reservationDateLabel setText: @""];
    [self.reservationStatusLabel setText: @""];
    [self.reservationTimeLabel setText: @""];
}

/*  Lays out the subviews and sets all appropriate values in the cell   */
- (void) layoutSubviews
{
    // set text values of the different reservation labels
    [self.reservationNameLabel setText: [self.reservation reservationDefectDescription]];
    [self.reservationStatusLabel setText: [self.reservation reservationStatusDescription]];
    [self.reservationTimeLabel setText: [self.reservation reservationDayPartDescription]];
    [self.reservationDateLabel setText: [self.reservation reservationDateDescription]];

    // configure the text labels
    [self.reservationDateLabel setNumberOfLines: 2];
    [self.reservationStatusLabel setFont: [UIFont systemFontOfSize: 15.0f]];
    [self.reservationDateLabel setFont: [UIFont preferredFontForTextStyle: UIFontTextStyleCaption1]];
    [self.reservationTimeLabel setFont: [UIFont preferredFontForTextStyle:UIFontTextStyleCaption2]];
    [self.reservationNameLabel setFont: [UIFont preferredFontForTextStyle: UIFontTextStyleSubheadline]];
}

/*  When the cell is being selected */
- (void)setSelected:(BOOL)selected animated:(BOOL)animated
{
    [super setSelected:selected animated:animated];
    // Configure the view for the selected state
}

#pragma mark    -
#pragma mark    KeyValue observing method

/*  When the status of the reservation changed this method gets called  */
- (void) observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary *)change context:(void
    *)context
{
    // respond to status change
    if ([keyPath isEqualToString: @"currentStatus"]) {

        // reset the reservation status label
        [self.reservationStatusLabel setText: [self.reservation reservationStatusDescription]];
        [self.reservationStatusLabel setTextColor: [UIColor darkGrayColor]];
    }
}

@end
```

```
//
//  SBReservationTableViewCell.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 24-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface SBReservationTableViewCell : UITableViewCell

// the label for the defect name and time description
@property (nonatomic, weak) IBOutlet UILabel *defectLabel;
@property (nonatomic, weak) IBOutlet UILabel *timeLabel;
@property (nonatomic, weak) IBOutlet UIActivityIndicatorView *activityIndicator;
@property (nonatomic, weak) IBOutlet UIImageView *crossImageView;

// the layer on the left of the cell
@property (nonatomic, strong) CALayer *lineLayer;

@end
```

```
//
//  SBReservationTableViewCell.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 24-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBReservationTableViewCell.h"
#import "SBColorManager.h"

@implementation SBReservationTableViewCell

/*  Initializer method from a storyboard instance   */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    // call super implementation
    self = [super initWithCoder: aDecoder];

    if (self) {

        // create the left custom layer
        self.lineLayer = [CALayer layer];
        [self.lineLayer setContentsScale: [[UIScreen mainScreen] scale]];
        [self.lineLayer setBackgroundColor: [[SBColorManager defaultColors] redColor].CGColor];
        [self.lineLayer setFrame: CGRectMake(0, 0, 10, self.frame.size.height + 1)];

        UIView *backgroundView = [[UIView alloc] initWithFrame: self.bounds];
        [backgroundView setBackgroundColor: [UIColor whiteColor]];
        [[backgroundView layer] addSublayer: self.lineLayer];

        // set the cells background view
        [self setBackgroundView: backgroundView];
    }

    return self;
}

/*  Respond to when the cell will be reused */
- (void) prepareForReuse
{
    // reset the text of both fields
    [self.timeLabel setText: @""];
    [self.defectLabel setText: @""];

    // set the image view start to not hidden and the activity indicator to not animating
    [self.activityIndicator stopAnimating];
    [self.crossImageView setHidden: NO];
}

@end
```

```
//
//  SBSelectableOptionTableViewCell.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 22-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface SBSelectableOptionTableViewCell : UITableViewCell

// the text indication of the cell
@property (nonatomic, weak) IBOutlet UILabel *optionLabel;

// the color used to draw the cell eft layer
@property (nonatomic, strong) UIColor *layerTintColor;

@end
```

```
//
//  SBSelectableOptionTableViewCell.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 22-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBSelectableOptionTableViewCell.h"
#import "SBColorManager.h"

@interface SBSelectableOptionTableViewCell()

// the custom layer with color on the left side of the cell
@property (nonatomic, strong) CALayer *lineLayer;

@end

@implementation SBSelectableOptionTableViewCell

/*  Initializer method for the table view cell  */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    // call super initializer
    self = [super initWithCoder: aDecoder];

    if (self) {

        // create the left custom layer
        self.lineLayer = [CALayer layer];
        [self.lineLayer setContentsScale: [[UIScreen mainScreen] scale]];
        [self.lineLayer setBackgroundColor: [[SBColorManager defaultColors] requestNavigationColor].CGColor];
        [self.lineLayer setFrame: CGRectMake(0, 0, 6, self.frame.size.height + 1)];

        UIView *backgroundView = [[UIView alloc] initWithFrame: self.bounds];
        [backgroundView setBackgroundColor: [UIColor whiteColor]];
        [[backgroundView layer] addSublayer: self.lineLayer];

        // set the cells background view
        [self setBackgroundView: backgroundView];
    }

    return self;
}

/*  Set the new layer color object  */
- (void) setLayerTintColor:(UIColor *)layerTintColor
{
    // set the new layer tint color
    _layerTintColor = layerTintColor;

    // append the layer color to the layer object
    [self.lineLayer setBackgroundColor: _layerTintColor.CGColor];
}

/*  When the cell will be reused reset the cell to original state   */
- (void) prepareForReuse
{
    // reset the option text
    [self.optionLabel setText: @""];
}

/*  When deallocated remove the layer from the cell */
- (void) dealloc
{
    // remove from super layer and set to nil
    [self.lineLayer removeFromSuperlayer];
    self.lineLayer = nil;
}

/*  Returns the cells selection style, which is always none */
- (UITableViewCellSelectionStyle) selectionStyle
{
    // return the none selection style
    return UITableViewCellSelectionStyleNone;
}

@end
```

```
//
//  SBTextField.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 18-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface SBTextField : UITextField

// the indexpath of this textfield in the corresponding tableview, also declares the position of the textfield in
    the tableview
@property (nonatomic, strong) NSIndexPath *correspondingIndexPath;
@property (nonatomic) UITableViewScrollPosition textfieldScrollPosition;

// the next textfield to show, can be nil
@property (nonatomic, weak) SBTextField *nextTextField;
@property (nonatomic, weak) SBTextField *previousTextField;

// the text indentation level in float
@property (nonatomic) CGFloat textIndentation;

// the string with information that is shown in the keyboard toolbar
@property (nonatomic, strong) NSString *toolbarInfo;
@property (nonatomic) BOOL isMarkedAsInvalid;

@end
```

```
//
//  SBTextField.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 18-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBTextField.h"
#import <QuartzCore/QuartzCore.h>

@implementation SBTextField

/*  Override the initializer to make sure the default of text indentation is set to 0   */
- (id) initWithFrame:(CGRect)frame
{
    // call super implementation
    self = [super initWithFrame: frame];

    if (self) {

        // set the default text indentation
        self.textIndentation = 0.0f;
    }

    return self;
}

/*  OVerride the is marked as invalid method    */
- (void) setIsMarkedAsInvalid:(BOOL)isMarkedAsInvalid
{
    // invert whether marked
    _isMarkedAsInvalid = isMarkedAsInvalid;

    // whentrue set the text color to red
    if (self.isMarkedAsInvalid) {
        [self setTextColor: [UIColor colorWithRed: 0.5859375 green: 0.1484375 blue: 0.1484375 alpha: 1]];
    } else {
        [self setTextColor: [UIColor blackColor]];
    }
}

/*  Override this method to set the text indentation for the textfield  */
- (CGRect) textRectForBounds:(CGRect) bounds
{
    // append the x coordinate of the bounds
    return CGRectInset(bounds, self.textIndentation, 0);
}

/*  OVerride this method to append insets for editing text  */
- (CGRect) editingRectForBounds:(CGRect) bounds
{
    // append the x coordinate of the bounds
    return CGRectInset(bounds, self.textIndentation, 0);
}

@end
```

```
//
//  SBToolBar.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 28-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface SBToolBar : UIToolbar
@end
```

```
//
//  SBToolBar.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 28-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBToolBar.h"

@implementation SBToolBar

/*  Initialize the toolbar from a decoder   */
- (id) initWithFrame:(CGRect)frame
{
    // call super class implementation
    self = [super initWithFrame: frame];

    if (self) {

        // this class is only used for the iOS 6 version of the application, in this version the tint color of the
            toolbar should be implicitly set to white to match the iOS 7 version
        [self setTintColor: [UIColor whiteColor]];
    }

    return self;
}

@end
```

```
//
//  SBViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 17-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>

#define kSBViewControllerTriangleIndicatorWidth     15
#define kSBViewControllerTriangleIndicatorHeight    8

@interface SBViewController : UIViewController

// the layer that represents the arrow in the view
@property (nonatomic, strong) CALayer *arrowLayer;

@end
```

```
//
//  SBViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 17-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBViewController.h"
#import "SBNavigationBarDetailView.h"
#import "SBColorManager.h"

@interface SBViewController ()

// the action that is executed when the back bar button is pressed
- (void) backBarbuttonItemPressed:(UIBarButtonItem *) sender;

@end

@implementation SBViewController

/*  When the view is loaded to additional view drawing  */
- (void) viewDidLoad
{
    // call super implementation
    [super viewDidLoad];

    // when the view is load create a new one for the arrow content
    self.arrowLayer = [[CALayer alloc] init];

    // configure the layer
    self.arrowLayer.delegate = self;
    self.arrowLayer.name = @"ArrowLayer";
    self.arrowLayer.frame = CGRectMake(0, 0, 320, kSBViewControllerTriangleIndicatorHeight);
    self.arrowLayer.backgroundColor = [UIColor clearColor].CGColor;
    self.arrowLayer.contentsScale = [[UIScreen mainScreen] scale];
    self.arrowLayer.zPosition = 10;

    // add the layer to the screen and redraw it
    [self.view.layer addSublayer: self.arrowLayer];
    [self.arrowLayer setNeedsDisplay];

    // set the default view background color
    [self.view setBackgroundColor: [[SBColorManager defaultColors] viewControllerBackgroundColor]];

    // add the title bar back button to the navigation bar when not the first view controller in the stack
    if (![[[self.navigationController viewControllers] objectAtIndex: 0] isEqual: self]) {

        // set the custom back bar button for this navigation item
        UIBarButtonItem *backBarButtonItem = [[UIBarButtonItem alloc] initWithImage: [UIImage imageNamed:
            @"TerugPijl"] style: UIBarButtonItemStylePlain target: self action: @selector(backBarbuttonItemPressed:)
            ];
        [backBarButtonItem setBackgroundVerticalPositionAdjustment: -kSBNavigationBarDetailViewHeight forBarMetrics:
            UIBarMetricsDefault];
        self.navigationItem.leftBarButtonItem = backBarButtonItem;
        [self.navigationItem setLeftItemsSupplementBackButton: NO];
    }

}

/*  Override the dealloc method to make sure to first remove the custom layer from its super layer  */
- (void) dealloc
{
    // remove the layer from its super layer and set it to nil
    [self.arrowLayer removeFromSuperlayer];
    self.arrowLayer = nil;
}

/*  Pops to the previous view controller in the navigation stack    */
- (void) backBarbuttonItemPressed:(UIBarButtonItem *)sender
{
    // pop to the previous view controller in the stack
    [self.navigationController popViewControllerAnimated: YES];
}

/*  When the view will start laying out its subviews    */
- (void) willMoveToParentViewController:(UIViewController *)parent
{
    // set display needs to be redrawn
    [self.view setNeedsDisplay];
}

/*  Override the draw layer method to make sure an arrow is drawn on the top of the view    */
- (void) drawLayer:(CALayer *) layer inContext:(CGContextRef) graphicsContext
{
    // only respond with drawing on the arrow layer
    if ([layer.name isEqualToString: @"ArrowLayer"]) {
```

```
        // set the new frame of the arrow layer
        layer.frame = CGRectMake(0, 0, 320, kSBViewControllerTriangleIndicatorHeight);

        // set fill and stroke color
        CGContextSetFillColor(graphicsContext, CGColorGetComponents([UIColor whiteColor].CGColor));
        CGContextSetStrokeColor(graphicsContext, CGColorGetComponents([UIColor whiteColor].CGColor));
        CGContextSetLineJoin(graphicsContext, kCGLineJoinRound);
        CGContextSetLineWidth(graphicsContext, 0.5);

        // allow antialiasing
        CGContextSetAllowsAntialiasing(graphicsContext, true);

        // set the fill color of the graphics context
        CGFloat leftTriangleCoordinate = (self.view.frame.size.width - kSBViewControllerTriangleIndicatorWidth) / 2;
        CGFloat rightTriangleCoordinate = (self.view.frame.size.width - leftTriangleCoordinate);

        // create mutable path
        CGMutablePathRef trianglePath = CGPathCreateMutable();
        CGPathMoveToPoint(trianglePath, NULL, leftTriangleCoordinate, 0);
        CGPathAddLineToPoint(trianglePath, NULL, rightTriangleCoordinate, 0);
        CGPathAddLineToPoint(trianglePath, NULL, (self.view.frame.size.width / 2),
            kSBViewControllerTriangleIndicatorHeight);
        CGPathCloseSubpath(trianglePath);

        // add the path to the context and fille the rect and borders with the provided colors
        CGContextAddPath(graphicsContext, trianglePath);
        CGContextFillPath(graphicsContext);
        CGContextAddPath(graphicsContext, trianglePath);
        CGContextStrokePath(graphicsContext);

        // release the path
        CGPathRelease(trianglePath);
    }
}

@end
```

```
//
//  SBAllReservationsViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 24-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBViewController.h"

@interface SBAllReservationsViewController : SBViewController <UITableViewDataSource, UITableViewDelegate,
    NSFetchedResultsControllerDelegate>

// the tableview holding all the reservations
@property (nonatomic, weak) IBOutlet UITableView *reservationTableView;

@end
```

```
//
//  SBAllReservationsViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 24-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBAllReservationsViewController.h"
#import "SBNavigationBar.h"
#import "SBNavigationBarDetailView.h"
#import "SBColorManager.h"
#import "SBAppDelegate.h"
#import "SBReservationTableViewCell.h"
#import "Reservation.h"
#import "SBDayPartDescriptor.h"
#import "Defect.h"
#import "Proceeding.h"
#import "SBReservationOverviewViewController.h"
#import "SBReservationManager.h"
#import "SBViewCompatabilityFactory.h"

@interface SBAllReservationsViewController ()

// the date formatter for the cell labels
@property (nonatomic, strong) NSDateFormatter *dateFormatter;

// a weak reference to the overview controller to get it notified about updates
@property (nonatomic, weak) SBReservationOverviewViewController *detailController;

// method shows or hides the view indicating that no reservation is currently available
- (void) setNoReservationDescriptionVisible:(BOOL) descriptionIsVisible;

// whether the description is visible for the no reservation
@property (nonatomic) BOOL isNoReservationDescriptionVisible;

@end

// the reuse identifier for these cells
static NSString *SBReservationTableViewCellReuseIdentifier  =   @"SBReservationCellReuseIdentifier";

@implementation SBAllReservationsViewController

/*  Initializes the view controller from the storyboard instance    */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    // call super implementation
    self = [super initWithCoder: aDecoder];

    if (self) {

        // create the date formatter
        self.dateFormatter = [[NSDateFormatter alloc] init];
        [self.dateFormatter setDateFormat: @"EEEE d MMMM"];
        [self.dateFormatter setLocale: [NSLocale currentLocale]];
    }

    return self;
}

/*  Reset objects when the view controller will be removed from memory  */
- (void) dealloc
{
    // reset the delegate of the fetched results controller
    [[SBReservationManager defaultReservationManager] setReservationControllerDelegate: nil];
}

/*  When the view information is loaded into memory this method is called and allows for interface initialization
    */
- (void) viewDidLoad
{
    // call super implementation
    [super viewDidLoad];

    // set the title of the navigation bar and the background color of the view
    [self setTitle: NSLocalizedString(@"all.viewcontroller.navigationbar.title", nil)];

    // set the delegate object self to the reservation manager fetched results controller
    [[SBReservationManager defaultReservationManager] setReservationControllerDelegate: self];

    // create the tableview which holds the information and register the corresponding xib file
    [self.reservationTableView registerNib: [UINib nibWithNibName: @"SBReservationTableViewCell" bundle: [NSBundle
        mainBundle]] forCellReuseIdentifier: SBReservationTableViewCellReuseIdentifier];
}

/*  When the view will be drawn on screen append the navigation bar detail view */
- (void) viewWillAppear:(BOOL) animated
```

```
{
    // hide tthe toolbar when visible
    [self.navigationController setToolbarHidden: YES animated: YES];

    // create the detail view with only the provided description and set it to the navigation bar
    SBNavigationBarDetailView *detailView = [[SBNavigationBarDetailView alloc] initWithDetailText: nil];

    // set the text corresponding to the current count of reservations
    NSInteger objectCount = [[[[SBReservationManager defaultReservationManager] fetchedResultsController] sections]
        objectAtIndex: 0] numberOfObjects];

    NSString *description = nil;

    // when no objects are available set the no reservation description visible
    if (objectCount == 0) {
        [self setNoReservationDescriptionVisible: YES];
        description = NSLocalizedString(@"all.viewcontroller.navigationbar.title.no.reservations", nil);
    } else {
        [self setNoReservationDescriptionVisible: NO];
        description = [NSString stringWithFormat:
            NSLocalizedString(@"all.viewcontroller.navigationbar.title.reservations", nil), objectCount];
    }

    // set the navigation title and tint color
    [detailView setDetailTitle: description];
    [detailView setDetailTintColor: [[SBColorManager defaultColors] redColor]];

    // add the view to the navigatio bar
    [(SBNavigationBar *)self.navigationController.navigationBar setDetailView: detailView];
}

/*  This method shows or hides the description that indicates no reservations are currently added by the user   */
- (void) setNoReservationDescriptionVisible:(BOOL) descriptionIsVisible
{
    // when the description should be shown check whether the view is not currently added to the view
    if (descriptionIsVisible && !self.isNoReservationDescriptionVisible) {

        // the font that will be used to draw the text
        UIFont *descriptionFont = [UIFont boldSystemFontOfSize: 11.0f];

        // calculate the used frame for the text that will be shown
        NSString *description = NSLocalizedString(@"all.viewcontroller.no.reservation.description", nil);
        CGSize descriptionFrame = [SBViewCompatabilityFactory boundingRectForString: description withSize:
            CGSizeMake(300, 200) options: NSStringDrawingUsesLineFragmentOrigin attributes: @{NSFontAttributeName:
            descriptionFont} context: nil];

        // create a label to hold the text
        UILabel *descriptionLabel = [[UILabel alloc] initWithFrame: CGRectZero];
        [descriptionLabel setText: description];
        [descriptionLabel setFrame: CGRectMake((self.view.frame.size.width - descriptionFrame.width) / 2, (self.view
            .frame.size.height - descriptionFrame.height) / 2, descriptionFrame.width, descriptionFrame.height)];
        [descriptionLabel setFont: descriptionFont];
        [descriptionLabel setNumberOfLines: 5];
        [descriptionLabel setTextAlignment: NSTextAlignmentCenter];
        [descriptionLabel setBackgroundColor: [UIColor clearColor]];

        // add the label to the screen
        [self.view addSubview: descriptionLabel];
        self.isNoReservationDescriptionVisible = YES;

        // hide the tableview
        [self.reservationTableView setHidden: YES];
    } else if (!descriptionIsVisible && self.isNoReservationDescriptionVisible) {

        // set the tableview to not hidden
        [self.reservationTableView setHidden: NO];

        // loop through the subviews to view the label and remove it
        for (UIView *subview in self.view.subviews) {

            // when a UILabel remove it
            if ([subview isKindOfClass: [UILabel class]]) {
                [subview removeFromSuperview];
            }
        }
    }
}

/*  When the application receives a memory warning this method is called    */
- (void) didReceiveMemoryWarning
{
    // call super implementation
    [super didReceiveMemoryWarning];
}

/*  Gets called when the next segue should be activated */
- (void) prepareForSegue:(UIStoryboardSegue *) segue sender:(id) sender
```

```
{
    // only prepare for the overview segue
    if ([[segue identifier] isEqualToString: @"userWantsReservationInfo:"]) {

        // get the selected reservation
        Reservation *selectedReservation = [[[SBReservationManager defaultReservationManager]
            fetchedResultsController ] objectAtIndexPath: [NSIndexPath indexPathForRow: [[self.reservationTableView
            indexPathForSelectedRow] section] inSection: 0]];

        // set the new detail view controller
        self.detailController = [segue destinationViewController];
        [self.detailController setReservation: selectedReservation];
    }
}

#pragma mark    -
#pragma mark    UITableView delegate & datasource

/*  Returns the amount of sections in the tableview, in other words the amount of objects   */
- (NSInteger) numberOfSectionsInTableView:(UITableView *) tableView
{
    // return the amount of objects in the reservation managers fetched results controller
    [[SBReservationManager defaultReservationManager] refreshReservationStatus];
    return [[[[SBReservationManager defaultReservationManager] fetchedResultsController] sections] objectAtIndex: 0
        ] numberOfObjects];
}

/*  Returns the amount of rows per section, this is always one  */
- (NSInteger) tableView:(UITableView *) tableView numberOfRowsInSection:(NSInteger) section
{
    return 1;
}

/*  Returns a configured cell for the provided indexpath    */
- (UITableViewCell *) tableView:(UITableView *) tableView cellForRowAtIndexPath:(NSIndexPath *) indexPath
{
    // dequeue a reusable cell from the tableview stack
    SBReservationTableViewCell *cell = (SBReservationTableViewCell *)[tableView dequeueReusableCellWithIdentifier:
        SBReservationTableViewCellReuseIdentifier];

    // get the current reservation object that needs to be shown in the cell
    Reservation *reservationForIndexPath = [[[SBReservationManager defaultReservationManager]
        fetchedResultsController] objectAtIndexPath: [NSIndexPath indexPathForRow: [indexPath section] inSection: 0]
        ];

    // set the cells time label and defect label
    [[cell timeLabel] setText: [NSString stringWithFormat: @"%@ %@", [[self.dateFormatter stringFromDate:
        [reservationForIndexPath selectedDate]] uppercaseString], [[reservationForIndexPath daypart] timeDescription
        ]]];
    [[cell defectLabel] setText: [[[[reservationForIndexPath proceedings] anyObject] defect] defectDescription]
        uppercaseString]];

    // the cell is not selectable
    [cell setSelectionStyle: UITableViewCellSelectionStyleNone];


    // when the reservation is being uploaded hide the cross image view and show the activity indicator
    if ([reservationForIndexPath isUploading]) {

        // hide the image view and start the animation
        [[cell activityIndicator] startAnimating];
        [[cell crossImageView] setHidden: YES];
    }

    return cell;
}

/*  Returns the default height for table view cells, which is 50    */
- (CGFloat) tableView:(UITableView *) tableView heightForRowAtIndexPath:(NSIndexPath *) indexPath
{
    return 50;
}

/*  Returns the height for the header provided in the method call   */
- (CGFloat) tableView:(UITableView *) tableView heightForHeaderInSection:(NSInteger) section
{
    // only the first section is higher than normal
    if (section == 0) {
        return 20.0;
    } else {
        return 3.0;
    }
}

/*  Returns the total height for the section footer provided    */
- (CGFloat) tableView:(UITableView *) tableView heightForFooterInSection:(NSInteger) section
{
```

```
    // only the last section has a higher footer
    if (section == ([self numberOfSectionsInTableView: tableView] - 1)) {
        return 20.0;
    } else {
        return 3.0;
    }
}

/*  When the cell at the provided indexpath is selected show the detail view for that reservation   */
- (void) tableView:(UITableView *) tableView didSelectRowAtIndexPath:(NSIndexPath *) indexPath
{
    // activate the next segue
    [self performSegueWithIdentifier: @"userWantsReservationInfo:" sender: self];
}

#pragma mark    -
#pragma mark    NSFetchedResultsController delegate

/*  Received from the fetched results controller when new updates are ready     */
- (void) controllerWillChangeContent:(NSFetchedResultsController *) controller
{
    // start the tableview update
    [self.reservationTableView beginUpdates];
}

/*  Gets called when the fetched results controller finished delivering the updates */
- (void) controllerDidChangeContent:(NSFetchedResultsController *) controller
{
    // stop the tableview updates
    [self.reservationTableView endUpdates];
}

/*  Gets called by the fetched results controller with information about a changed object at provided index path and
    the change type    */
- (void) controller:(NSFetchedResultsController *) controller didChangeObject:(id) anObject atIndexPath:(NSIndexPath
    *) indexPath forChangeType:(NSFetchedResultsChangeType) type newIndexPath:(NSIndexPath *) newIndexPath
{
    // when an object is updated and the sub view controller is visible update that object when needed
    if (self.detailController != nil && [[self.detailController reservation] isEqual: anObject] && type ==
        NSFetchedResultsChangeUpdate) {
        [self.detailController refreshReservation];
    }

    // when the detail view is visible but being deleted pop to previous
    if (self.detailController != nil && [[self.detailController reservation] isEqual: anObject] && type ==
        NSFetchedResultsChangeDelete) {
        [self.navigationController popViewControllerAnimated: YES];
    }

    // when an insert or delete action should be performed check whether 0 objects are shown to show the no
        reservation description
    if (type == NSFetchedResultsChangeDelete || type == NSFetchedResultsChangeInsert) {

        // when objects are available show the description else hide it
        if ([[[[SBReservationManager defaultReservationManager] fetchedResultsController] sections] objectAtIndex:
            0] numberOfObjects] == 0) {
            [self setNoReservationDescriptionVisible: NO];
        } else {
            [self setNoReservationDescriptionVisible: YES];
        }
    }

    // switch to perform the correct action for each update of the fetched results controller
    switch (type) {

        // perform action when a reservation is deleted from the store
        case NSFetchedResultsChangeDelete:

            // delete the section at the provided indexpath's row with an automatic row animation
            [self.reservationTableView deleteSections: [NSIndexSet indexSetWithIndex: [indexPath row]]
                withRowAnimation: UITableViewRowAnimationAutomatic];
            break;

        // perform action when a new reservation object is inserted in the store
        case NSFetchedResultsChangeInsert:

            // insert a new section at the indexpath's row with automatic row animation
            [self.reservationTableView insertSections: [NSIndexSet indexSetWithIndex: [indexPath row]]
                withRowAnimation: UITableViewRowAnimationAutomatic];
            break;

        // perform action when a reservation is moved from indexpath to a new indexpath
        case NSFetchedResultsChangeMove:

            // move the two section of the given indexpath's row's
            [self.reservationTableView moveSection: [indexPath row] toSection: [newIndexPath row]];
            break;
```

```
        // when a reservation is updated update the cell representing this object
        case NSFetchedResultsChangeUpdate:

            // update the cell at given section
            [self.reservationTableView reloadSections: [NSIndexSet indexSetWithIndex: [indexPath row]]
                withRowAnimation: UITableViewRowAnimationAutomatic];
            break;
    }
}

/*  gets called when a section in the fetched results controller changes, we do not respond to this method  */
- (void) controller:(NSFetchedResultsController *) controller didChangeSection:(id<NSFetchedResultsSectionInfo>)
    sectionInfo atIndex:(NSUInteger) sectionIndex forChangeType:(NSFetchedResultsChangeType) type{}

@end
```

```
//
//  SBAppointmentViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBViewController.h"
#import "SBOperationDelegate.h"
#import "SBURLOperationDelegate.h"

@interface SBAppointmentViewController : SBViewController <UITableViewDataSource, UITableViewDelegate,
    SBOperationDelegate, SBURLOperationDelegate, UIPickerViewDataSource, UIPickerViewDelegate>

// the tableview with date content selection
@property (nonatomic, weak) IBOutlet UITableView *dateTableView;

@end
```

```
//
//  SBAppointmentViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBAppointmentViewController.h"
#import "SBNavigationBar.h"
#import "SBNavigationBarDetailView.h"
#import "SBColorManager.h"
#import "SBAPIRequest.h"
#import "SBURLOperation.h"
#import "SBOperationManager.h"
#import "SBAPIResponse.h"
#import "SBAvailableDateDescriptor.h"
#import "SBDayPartDescriptor.h"
#import "SBReservationManager.h"
#import "Reservation.h"
#import "SBReservationOverviewViewController.h"
#import "SBViewCompatabilityFactory.h"

@interface SBAppointmentViewController ()

// when the done button is pressed this method will be called
- (void) finishButtonPressed:(UIBarButtonItem *) sender;

// refreshes all content on the page
- (void) reloadContent;

// parses the received dictionary to available date options
- (void) parseReceivedOptions:(NSArray *) optionArray;

// properties for the selected row in the first and second component
@property (nonatomic) NSInteger selectedRowInFirstComponent;
@property (nonatomic) NSInteger selectedRowInSecondComponent;

// whether the date selector is visible to the user
@property (nonatomic) BOOL isDateSelectionViewVisible;
@property (nonatomic) BOOL userHasSelectedOption;

// whether the date options are currently being loaded
@property (nonatomic) BOOL isLoadingDateOptions;

// the array of available dates
@property (nonatomic, strong) NSArray *options;

// the date picker view and date formatter for the first component in the date picker
@property (nonatomic, strong) UIPickerView *datePicker;
@property (nonatomic, strong) NSDateFormatter *dateFormatter;

@end

// the name of the operation that will be started
static NSString *SBDateRefreshOperationName     =   @"SBOperationDateRefresh";

// the message provided by the API response when no dates are available
static NSString *SBDateRefreshErrorNoDates      =   @"api.rest.available_dates.none";

@implementation SBAppointmentViewController

/*  initializer method which gets called from the storyboard instance   */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    // call super initializer
    self = [super initWithCoder: aDecoder];

    if (self) {

        // set default values]
        self.isDateSelectionViewVisible = NO;
        self.isLoadingDateOptions = NO;
        self.userHasSelectedOption = NO;

        // create the date formatter
        self.dateFormatter = [[NSDateFormatter alloc] init];
        [self.dateFormatter setDateFormat: @"EEEE d MMMM"];
        //[self.dateFormatter setLocale: [NSLocale localeWithLocaleIdentifier: @"nl_NL"]];
        [self.dateFormatter setLocale: [NSLocale currentLocale]];
    }

    return self;
}

/*  When the view content is loaded into memory this method gets called and allows for user interface initialization
    */
```

```
- (void) viewDidLoad
{
    // call super implementation
    [super viewDidLoad];

    // set the title of the new view controller
    [self setTitle: NSLocalizedString(@"location.viewcontroller.title", nil)];

    // create the button for the toolbar
    UIButton *finishButton = [[UIButton alloc] initWithFrame: CGRectMake(160, 0, 160, 44)];
    [finishButton addTarget: self action: @selector(finishButtonPressed:) forControlEvents:
        UIControlEventTouchUpInside];
    [finishButton setBackgroundColor: [[SBColorManager defaultColors] requestNavigationColor]];
    [[finishButton titleLabel] setFont: [UIFont systemFontOfSize: 13.0f]];
    [finishButton setTitle: NSLocalizedString(@"appointment.viewcontroller.toolbar.finish.title", nil) forState:
        UIControlStateNormal];

    // create the bar button for the finish button
    UIBarButtonItem *finishBarButtonItem = [[UIBarButtonItem alloc] initWithCustomView: finishButton];
    UIBarButtonItem *fixedSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
        UIBarButtonSystemItemFixedSpace target: nil action: nil];
    fixedSpace.width -= 16.0f;

    // set the new toolbar items
    [self setToolbarItems: [self.toolbarItems arrayByAddingObjectsFromArray: [NSArray arrayWithObjects:
        finishBarButtonItem, fixedSpace, nil]] animated: YES];

    // set the tableview background color
    [self reloadContent];

    // everytime this view controller is loaded into memory the date choices previously set should be reset
    [[[SBReservationManager defaultReservationManager] currentReservation] setSelectedDate: nil];
    [[[SBReservationManager defaultReservationManager] currentReservation] setDaypart: nil];
}

/*  when the view will appear on screen show the new detail view    */
- (void) viewWillAppear:(BOOL) animated
{
    // call super implementation
    [super viewWillAppear: animated];

    // create the new detail view for the navigation bar
    SBNavigationBarDetailView *detailView = [[SBNavigationBarDetailView alloc] initWithDetailText: NSLocalizedString
        (@"appointment.viewcontroller.navigation.detail.description", nil)];

    // configure the details
    [detailView setDetailTitle: NSLocalizedString(@"appointment.viewcontroller.navigation.detail.title", nil)];
    [detailView setDetailTintColor: [[SBColorManager defaultColors] requestNavigationColor]];
    [detailView setDetailImage: [UIImage imageNamed: @"AfspraakStap"]];

    // set the new navigation bar detail view
    [(SBNavigationBar *)self.navigationController.navigationBar setDetailView: detailView];
}

/*  When a memory warning is received try to free up resources  */
- (void) didReceiveMemoryWarning
{
    // call super implementation
    [super didReceiveMemoryWarning];
}

/*  When the finish button is pressed this method will activate the next segue when able to */
- (void) finishButtonPressed:(UIBarButtonItem *) sender
{
    // when options could be selected and the user has selected a date
    if ([self.options count] > 0 && self.userHasSelectedOption) {
        [self performSegueWithIdentifier: @"userDidSelectAppointmentDate:" sender: self];
    }
}

/*  starts to reload the date content   */
- (void) reloadContent
{
    // when not currently reloading the content continue else return
    if (self.isLoadingDateOptions) {
        return;
    }

    // create the API request object
    SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPIAvailableDatesRequestType timeout: 60
        ];

    // create the operation and set its delegate object
    NSDictionary *operationInfo = @{SBOperationNameOptionKey: SBDateRefreshOperationName};
    SBURLOperation *dateOperation = [[SBURLOperation alloc] initWithAPIRequest: APIRequest options: operationInfo];
    [dateOperation setDelegate: self];
```

```
    // start the operation
    [[SBOperationManager defaultManager] addOperation: dateOperation];

    // set that the date selection cell can not be shown and the dates are being reloaded
    self.isLoadingDateOptions = YES;
    self.isDateSelectionViewVisible = NO;

    // reload the first section
    [self.dateTableView reloadSections: [NSIndexSet indexSetWithIndex: 0] withRowAnimation:
        UITableViewRowAnimationAutomatic];
}

/*  Parse the new received date objects into an array   */
- (void) parseReceivedOptions:(NSArray *) optionArray
{
    // create the container for the next date descriptor and mutable array to hold the objects
    SBAvailableDateDescriptor *dateDescriptor = nil;
    NSMutableArray *dateObjects = [NSMutableArray array];

    // pasre the dates and add them to the possible dates array
    for (int i = 0; i < [optionArray count]; i++) {

        // create the new date descriptor from the current dictionary and add it to the mutable container
        dateDescriptor = [[SBAvailableDateDescriptor alloc] initWithDictionary: [optionArray objectAtIndex: i]];
        [dateObjects addObject: dateDescriptor];
    }

    // create the actual options array from the mutable array
    self.options = [NSArray arrayWithArray: dateObjects];
}

/*  Returns the initialized date picker view object */
- (UIPickerView *) datePicker
{
    // when the date picker is already initialized return it
    if (_datePicker != nil) {
        return _datePicker;
    }

    // retrieve a cell to find out its bounds
    CGRect actualBounds = [[[self.dateTableView cellForRowAtIndexPath: [NSIndexPath indexPathForRow: 0 inSection: 0]
        ] contentView] bounds];

    // date picker is not created yet so created it ourself
    _datePicker = [[UIPickerView alloc] initWithFrame: CGRectMake(0, 0, actualBounds.size.width, 0)];

    // configure the date picker
    [_datePicker setDelegate: self];
    [_datePicker setDataSource: self];

    // return the picker
    return _datePicker;
}

/*  When the view controller will prepare for the next segue save and prepare the next view controller  */
- (void) prepareForSegue:(UIStoryboardSegue *) segue sender:(id) sender
{
    // when the next view controller is the overview controller
    if ([[segue identifier] isEqualToString: @"userDidSelectAppointmentDate:"]) {

        // get the current reservation and set the date and daypart object
        Reservation *currentReservation = [[SBReservationManager defaultReservationManager] currentReservation];
        [currentReservation setSelectedDate: [[self.options objectAtIndex: self.selectedRowInFirstComponent]
            availableDate]];
        [currentReservation setDaypart: [[[self.options objectAtIndex: self.selectedRowInFirstComponent]
            availableTime] objectAtIndex: self.selectedRowInSecondComponent]];

        // add the reservation object to the next view controller
        [[segue destinationViewController] setReservationRequiresConfirmation: YES];
        [[segue destinationViewController] setReservation: currentReservation];
    }
}

#pragma mark    -
#pragma mark    UITableView delegate & datasource

/*  Returns the amount of sections in the tableview */
- (NSInteger) numberOfSectionsInTableView:(UITableView *) tableView
{
    // this tableview only has one section
    return 1;
}

/*  Returns the amount of rows currently in the section */
- (NSInteger) tableView:(UITableView *) tableView numberOfRowsInSection:(NSInteger) section
{
    // when the date selection is visible return 2 rows, otherwise one
```

```
    if (self.isDateSelectionViewVisible) {
        return 2;
    } else {
        return 1;
    }
}

/*  Returns the cell appropriate for the indexpath provided */
- (UITableViewCell *) tableView:(UITableView *) tableView cellForRowAtIndexPath:(NSIndexPath *) indexPath
{
    // create a new cell
    UITableViewCell *cell = [[UITableViewCell alloc] initWithStyle: UITableViewCellStyleDefault reuseIdentifier: nil
        ];

    // for the first row in the first section check what to show, activity indicator are the current selected date
    if ([indexPath section] == 0 && [indexPath row] == 0) {

        // when currently loading the date options show activity indicator
        if (self.isLoadingDateOptions) {

            // create the activity indicator and set its frame and center coordinate
            UIActivityIndicatorView *activityIndicator = [[UIActivityIndicatorView alloc]
                initWithActivityIndicatorStyle: UIActivityIndicatorViewStyleGray];
            [activityIndicator setFrame:CGRectMake(0, 0, 44, 44)];
            [activityIndicator setCenter: CGPointMake(22, 22)];
            [activityIndicator setBackgroundColor: [UIColor clearColor]];

            // start the animation and add to the subview
            [cell.contentView addSubview: activityIndicator];
            [activityIndicator startAnimating];

            // set the description for loading
            UILabel *descriptionLabel = [[UILabel alloc] initWithFrame: CGRectMake(activityIndicator.frame.size.
                width, 0, (300 - activityIndicator.frame.size.width), 44)];
            [descriptionLabel setFont: [UIFont italicSystemFontOfSize: 13]];
            [descriptionLabel setText: NSLocalizedString(@"appointment.viewcontroller.cell.loading.description", nil
                )];
            [descriptionLabel setBackgroundColor: [UIColor clearColor]];

             // add the label to the cell
            [cell.contentView addSubview: descriptionLabel];

        } else {

            // set text font
            [[cell textLabel] setFont: [UIFont boldSystemFontOfSize: 11]];

            // set the cell title
            if ([self.options count] > 0 && self.userHasSelectedOption) {

                // set the date and day part description in the first cell
                [[cell textLabel] setText: [NSString stringWithFormat: @"%@ %@", [[self.dateFormatter
                    stringFromDate: [[self.options objectAtIndex: self.selectedRowInFirstComponent] availableDate]]
                    uppercaseString], [[[[self.options objectAtIndex: self.selectedRowInFirstComponent]
                    availableTime] objectAtIndex: self.selectedRowInSecondComponent] timeDescription]]];

            } else if ([self.options count] > 0) {
                [[cell textLabel] setText: NSLocalizedString(@"appointment.viewcontroller.cell.noselection", nil)];
            } else {

                // set the title that no dates are available
                [[cell textLabel] setText: NSLocalizedString(@"appointment.viewcontroller.alert.noavailable.title",
                    nil)];
            }

            // only show the dropdown image when options are available
            if ([self.options count] > 0) {

                // create image view for the selection indicator
                UIImageView *dropdownImage = [[UIImageView alloc] initWithImage: [UIImage imageNamed:
                    @"AfspraakMakenPijl"]];
                [dropdownImage setFrame: CGRectMake((290 - dropdownImage.image.size.width), (44 - dropdownImage.
                    image.size.height) / 2, dropdownImage.image.size.width, dropdownImage.image.size.height)];

                // add image view to the cell
                [cell addSubview: dropdownImage];
            }
        }
    } else if ([indexPath section] == 0 && [indexPath row] == 1) {

        // add the date picker to the cell as subview
        [cell.contentView addSubview: [self datePicker]];
    }

    // create borders around the cell
    cell.layer.masksToBounds = YES;
    cell.layer.cornerRadius = 3.0f;
```

```
    // set a nil background view
    [cell setBackgroundView: nil];

    // the cell shows no selection indicator
    [cell setSelectionStyle: UITableViewCellSelectionStyleNone];

    return cell;
}

/*  Respond to when a tableview cell has been tapped by the user, only respond to the first cell    */
- (void) tableView:(UITableView *) tableView didSelectRowAtIndexPath:(NSIndexPath *) indexPath
{
    // only check for index row one
    if ([indexPath section] == 0 && [indexPath row] == 0 && [self.options count] > 0) {

        // toggle the view visible and reload the section
        self.isDateSelectionViewVisible = !self.isDateSelectionViewVisible;
        [self.dateTableView reloadSections: [NSIndexSet indexSetWithIndex: 0] withRowAnimation:
            UITableViewRowAnimationAutomatic];
    }
}

/*  Returns the height for a cell at the given indexpath    */
- (CGFloat) tableView:(UITableView *) tableView heightForRowAtIndexPath:(NSIndexPath *) indexPath
{
    // the first row is standard height, otherwise return height of UIPickerview
    if ([indexPath row] == 0) {
        return 44;
    } else {
        // return picker height
        return 216;
    }
}

#pragma mark    -
#pragma mark    UIPickerView delegate & datasource

/*  Returns the amount of components in this pickerview, defaults to 2. One for the date and one for the day part
    */
- (NSInteger) numberOfComponentsInPickerView:(UIPickerView *) pickerView
{
    return 2;
}

/*  Returns the number of rows in the selected date of the first component  */
- (NSInteger) pickerView:(UIPickerView *) pickerView numberOfRowsInComponent:(NSInteger) component
{
    // for the first component return the amount of options
    if (component == 0) {
        return [self.options count];
    } else {

        // get the selected row in the first component and return the amount of dayparts available in that objects
            date
        return [[[self.options objectAtIndex: [pickerView selectedRowInComponent: 0]] availableTime] count];
    }
}

/*  when the picker view selected row in the provided component this method should respond to it    */
- (void) pickerView:(UIPickerView *) pickerView didSelectRow:(NSInteger) row inComponent:(NSInteger) component
{
    // when this is the first time the user has selected a row toggle the boolean
    if (!self.userHasSelectedOption) {
        self.userHasSelectedOption = YES;
    }

    // when the first component changed the second component should be reloaded
    if (component == 0) {

        // reload and set the selected component
        [pickerView reloadComponent: 1];
        self.selectedRowInFirstComponent = row;

        // when the selected row in the second component is not already 0, animate it to that position
        if (self.selectedRowInSecondComponent != 0) {

            // animate to the first row and reset
            [pickerView selectRow: 0 inComponent: 1 animated: YES];
            self.selectedRowInSecondComponent = 0;
        }
    } else if (component == 1) {

        // set the selected row in the second component
        self.selectedRowInSecondComponent = row;
    }
```

```
    // reload the first cell in the tableview
    [self.dateTableView reloadRowsAtIndexPaths: [NSArray arrayWithObject: [NSIndexPath indexPathForRow: 0 inSection:
        0]] withRowAnimation: UITableViewRowAnimationNone];
}

/*  Returns the height of the component provided    */
- (CGFloat) pickerView:(UIPickerView *) pickerView widthForComponent:(NSInteger)component
{
    // get the width of a cell
    CGFloat frameWidth = [[self.dateTableView cellForRowAtIndexPath: [NSIndexPath indexPathForRow: 0 inSection: 0]]
        bounds].size.width;

    // when on iOS 6 the picker view widht is smaller than it actual is
    if ([SBViewCompatabilityFactory cocoaVersion] == UICocoaVersionIdentifier_iOS_6_0) {
        frameWidth -= 40;
    }

    // the first component is wider than the second one
    if (component == 0) {
        return (frameWidth * 0.666667);
    } else {
        return (frameWidth * 0.333334);
    }
}

/*  Returns a view object that will be displayed, can reuse the view when provided  */
- (UIView *) pickerView:(UIPickerView *) pickerView viewForRow:(NSInteger) row forComponent:(NSInteger) component
    reusingView:(UIView *) view
{
    // create default label which will be initialized or reused
    UILabel *textLabel = nil;

    // when no view is provided create a new view
    if (view == nil) {

        // create a new label view with a frame prepared for the current component
        textLabel = [[UILabel alloc] initWithFrame: CGRectMake(0, 0, [self pickerView: pickerView widthForComponent:
            component], 20)];

        // set the font depending on the component
        if (component == 0) {
            [textLabel setFont: [UIFont boldSystemFontOfSize: 13.0f]];
        } else {
            [textLabel setFont: [UIFont boldSystemFontOfSize: 12.0f]];
        }

        // text align always in center
        [textLabel setTextAlignment: NSTextAlignmentCenter];
        [textLabel setBackgroundColor: [UIColor clearColor]];

    } else {

        // reuse the view
        textLabel = (UILabel *) view;
    }

    // set the text of the label depending on the component
    if (component == 0) {
        [textLabel setText: [[self.dateFormatter stringFromDate: [[self.options objectAtIndex: row] availableDate]]
            uppercaseString]];
    } else {
        [textLabel setText: [[[[self.options objectAtIndex: [pickerView selectedRowInComponent: 0]] availableTime]
            objectAtIndex: row] timeDescription]];
    }

    // return the reusable view
    return textLabel;
}

#pragma mark    -
#pragma mark    SBOperation delegate

/*  When the operation finished execution with an error this method gets called with the provided error */
- (void) operation:(SBOperation *)operation didFailWithError:(NSError *)error
{
    // an internel server error occured
    UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
        NSLocalizedString(@"appointment.viewcontroller.alert.loading.title", nil) message: [error
        localizedDescription] delegate: self cancelButtonTitle:
        NSLocalizedString(@"appointment.viewcontroller.alert.noaction.title", nil) otherButtonTitles:
        NSLocalizedString(@"appointment.viewcontroller.alert.retry.title", nil), nil];

    // show the alert
    [alertView show];
}

/*  When the operation finished execution this method gets called   */
```

```
- (void) operation:(SBOperation *)operation didFinishWithResult:(SBAPIResponse *)response
{
    // check for the refresh dates operation has finished
    if ([[[operation options] valueForKey: SBOperationNameOptionKey] isEqualToString: SBDateRefreshOperationName]) {

        // check for an error
        if ([response objectResultType] == SBAPIResultSuccessType) {

            // data is successfully loaded, parse it
            [self parseReceivedOptions: (NSArray *)[response resultDictionary]];

        } else if ([response objectResultType] == SBAPIResultErrorType && [[response resultMessage] isEqualToString:
            SBDateRefreshErrorNoDates]) {

            // an error occured because there are no dates available to be selected, show alert
            UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
                NSLocalizedString(@"appointment.viewcontroller.alert.noavailable.title", nil) message:
                NSLocalizedString(@"appointment.viewcontroller.alert.noselection.description", nil) delegate: nil
                cancelButtonTitle: nil otherButtonTitles:
                NSLocalizedString(@"appointment.viewcontroller.alert.noaction.title", nil), nil];

            // show the alert
            [alertView show];

        } else {

            // an internel server error occured
            UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
                NSLocalizedString(@"appointment.viewcontroller.alert.loading.title", nil) message: NSLocalizedString
                (@"appointment.viewcontroller.alert.internal.description", nil) delegate: self cancelButtonTitle:
                NSLocalizedString(@"appointment.viewcontroller.alert.noaction.title", nil) otherButtonTitles:
                NSLocalizedString(@"appointment.viewcontroller.alert.retry.title", nil), nil];

            // show the alert
            [alertView show];
        }
    }

    // set new boolean values
    self.isLoadingDateOptions = NO;
    [self.dateTableView reloadSections: [NSIndexSet indexSetWithIndex: 0] withRowAnimation:
        UITableViewRowAnimationAutomatic];
}

@end
```

```
//
//  SBDefectSelectionViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 22-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBViewController.h"

@interface SBDefectSelectionViewController : SBViewController <UITableViewDataSource, UITableViewDelegate>

// the selected element object id from the previous step
@property (nonatomic, strong) NSManagedObjectID *elementIdentifier;

// the tableview which holds the final defect choice
@property (nonatomic, weak) IBOutlet UITableView *defectTableView;

@end
```

```
//
//  SBDefectSelectionViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 22-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBDefectSelectionViewController.h"
#import "SBNavigationBar.h"
#import "SBNavigationBarDetailView.h"
#import "SBAppDelegate.h"
#import "SBSelectableOptionTableViewCell.h"
#import "SBReservationManager.h"
#import "SBColorManager.h"

@interface SBDefectSelectionViewController ()

// fetches all objects that can be selected by the user from the CoreData store
- (void) retrieveSelectableObjects;

// indicates that the view should be reloaded and refresh its content
- (void) refreshContent;
- (void) showActivityView;

// the main managed object context for the application
@property (nonatomic, weak) NSManagedObjectContext *mainContext;

// the array of fetched defects Object ID's
@property (nonatomic, strong) NSArray *defects;

// the activity indicator view indicating read activity
@property (nonatomic, strong) UIActivityIndicatorView *activityIndicator;

@end

// the static reuse idnetifier for defect cells
static NSString *SBDefectSelectionCellReuseIdentifier   =   @"SBDefectSelectionReuseIdentifier";

@implementation SBDefectSelectionViewController

/*  Initializer method which gets called from the storyboard    */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    // call super initializer
    self = [super initWithCoder: aDecoder];

    if (self) {

        // set the main managed object context of the application
        self.mainContext = [(SBAppDelegate *)[[UIApplication sharedApplication] delegate] managedObjectContext];
    }

    return self;
}

/*  When the view is loaded in memory do initialization for interface in this method    */
- (void) viewDidLoad
{
    [super viewDidLoad];

    // set the title of the controller
    [self setTitle: NSLocalizedString(@"location.viewcontroller.title", nil)];

    // set the nib used for reusing cells
    [self.defectTableView registerNib: [UINib nibWithNibName: @"SBSelectableOptionTableViewCell" bundle: [NSBundle
        mainBundle]] forCellReuseIdentifier: SBDefectSelectionCellReuseIdentifier];

    // start loading the content
    [self retrieveSelectableObjects];
}

/*  When the view will appear on screen change frames and navigation bar detail view    */
- (void) viewWillAppear:(BOOL) animated
{
    // set the toolbar to hidden when shown
    [self.navigationController setToolbarHidden: YES animated: YES];

    // create the new detail view
    SBNavigationBarDetailView *detailView = [[SBNavigationBarDetailView alloc] initWithDetailText: NSLocalizedString
        (@"defect.viewcontroller.navigation.description", nil)];

    // configure the detail view
    [detailView setDetailTitle: NSLocalizedString(@"defect.viewcontroller.navigation.detail.title", nil)];
    [detailView setDetailTintColor: [UIColor colorWithRed: 0.54296875 green: 0.8359375 blue: 0.73828125 alpha: 1.0]]
        ;
    [detailView setDetailImage: [UIImage imageNamed: @"DefectStap"]];
```

```
    // append the detail view to the navigation bar
    [(SBNavigationBar *)self.navigationController.navigationBar setDetailView: detailView];
}

/*  When a memory warning is received try to clean up some resources    */
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

/*  Start to load the objects to show as selectable */
- (void) retrieveSelectableObjects
{
    // create the activity indicator to show that the data is being loaded
    [self showActivityView];

    // create a new managed object context to load the data from
    NSManagedObjectContext *managedObjectContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:
        NSPrivateQueueConcurrencyType];
    [managedObjectContext setPersistentStoreCoordinator: [(SBAppDelegate *)[[UIApplication sharedApplication]
        delegate] persistentStoreCoordinator]];

    // perform the block that loads the data
    [managedObjectContext performBlock: ^{

        // get the managed object that was selected in the previous view controller
        NSManagedObject *previousManagedObject = [self.mainContext objectWithID: self.elementIdentifier];

        // create the fetch request for objects to retrieve
        NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] initWithEntityName: @"Defect"];
        [fetchRequest setPredicate: [NSPredicate predicateWithFormat: @"element.elementId = %@",
            [previousManagedObject valueForKeyPath: @"elementId"]]];

        // create and add the sort descriptor
        NSSortDescriptor *sortDescriptor = [NSSortDescriptor sortDescriptorWithKey: @"defectDescription" ascending:
            YES];
        [fetchRequest setSortDescriptors: [NSArray arrayWithObject: sortDescriptor]];

        // create error object and lock the managed object context
        NSError *error = nil;
        [managedObjectContext lock];

        // execute the fetch request
        NSArray *results = [managedObjectContext executeFetchRequest: fetchRequest error: &error];

        // unlock the context
        [managedObjectContext unlock];

        // when no error occured set the rooms array
        if (error == nil) {

            // create a new array with all object ID's fetched in the background
            NSMutableArray *objectIDs = [NSMutableArray array];

            // add each object ID to the array
            for (NSManagedObject *managedObject in results) {
                [objectIDs addObject: [managedObject objectID]];
            }

            // set the new array and perform the refreshing of the view on the main thread
            self.defects = [NSArray arrayWithArray: objectIDs];
            [self performSelectorOnMainThread: @selector(refreshContent) withObject: nil waitUntilDone: NO];
        }
    }];
}

/*  Gets called on the main thread and indicates that new data is available to be shown */
- (void) refreshContent
{
    // set the tableview to not be hidden
    [self.defectTableView setHidden: NO];

    // remove the activity indicator when it is added to the view
    if ([self.activityIndicator superview] != nil) {

        // stop animation and remove from memory
        [self.activityIndicator stopAnimating];
        [self.activityIndicator removeFromSuperview];
        self.activityIndicator = nil;
    }

    // reload table data
    [self.defectTableView reloadData];
}
```

```
/*  When the data is being loaded from CoreData this method shows the activity indicator    */
- (void) showActivityView
{
    // hide the tableview
    [self.defectTableView setHidden: YES];

    // create and configure the activity indicator view
    self.activityIndicator = [[UIActivityIndicatorView alloc] initWithActivityIndicatorStyle:
        UIActivityIndicatorViewStyleGray];
    [self.activityIndicator setFrame: self.view.bounds];
    [self.activityIndicator setCenter: CGPointMake((self.view.frame.size.width / 2), (self.view.frame.size.height /
        2))];

    // add to the view and start the animation
    [self.view addSubview: self.activityIndicator];
    [self.activityIndicator startAnimating];
}

#pragma mark    -
#pragma mark    UITableView datasource & delegate

/*  Returns the amount of sections in the tableview, each section represents a selectable option    */
- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView
{
    // return the amount of objects in the element array
    return [self.defects count];
}

/*  Returns the amount of rows per section, in this case always one */
- (NSInteger) tableView:(UITableView *) tableView numberOfRowsInSection:(NSInteger) section
{
    return 1;
}

/*  Returns the configured cell for the given indexpath */
- (UITableViewCell *) tableView:(UITableView *) tableView cellForRowAtIndexPath:(NSIndexPath *) indexPath
{
    // dequeue a cell from the tableview, is always an object because the nib is registered in view did load
    SBSelectableOptionTableViewCell *cell = (SBSelectableOptionTableViewCell *)[tableView
        dequeueReusableCellWithIdentifier: SBDefectSelectionCellReuseIdentifier];

    // get the corresponding object for this indexpath
    NSManagedObject *currentObject = (NSManagedObject *)[self.mainContext objectWithID: [self.defects objectAtIndex:
        [indexPath section]]];

    // set the text label of the cell
    [[cell optionLabel] setText: [[currentObject valueForKeyPath: @"defectDescription"] uppercaseString]];
    [cell setLayerTintColor: [[SBColorManager defaultColors] requestNavigationColor]];

    return cell;
}

/*  Returns the height of the header views in the tableview */
- (CGFloat) tableView:(UITableView *) tableView heightForHeaderInSection:(NSInteger) section
{
    // only the first section has a higher header
    if (section == 0) {
        return 20;
    } else {
        return 4;
    }
}

/*  Return the appended footer height for all cells */
- (CGFloat) tableView:(UITableView *) tableView heightForFooterInSection:(NSInteger) section
{
    // only the last cell has a higher footer for the white space
    if (section == ([self.defects count] - 1)) {
        return 10;
    } else {
        return 4;
    }
}

/*  Gets notified when the user has selected a table view cell, should show the next segue  */
- (void) tableView:(UITableView *) tableView didSelectRowAtIndexPath:(NSIndexPath *) indexPath
{
    // perform the next segue
    [self performSegueWithIdentifier: @"userDidSelectDefect:" sender: self];

    // get the current selected managed object and save the selected value
    NSManagedObject *managedObject = [self.mainContext objectWithID: [self.defects objectAtIndex: [indexPath section
        ]]];
    [[SBReservationManager defaultReservationManager] processValue: managedObject forCurrentProceedingPropertyKey:
        @"defect"];
}
```

```
@end
```

```
//
//  SBElementSelectionViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 21-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBViewController.h"

@interface SBElementSelectionViewController : SBViewController <UITableViewDataSource, UITableViewDelegate>

// the selected identifier for the space
@property (nonatomic, strong) NSManagedObjectID *spaceIdentifier;

// the tableview which holds all element choices
@property (nonatomic, weak) IBOutlet UITableView *elementTableView;

@end
```

```
//
//  SBElementSelectionViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 21-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBElementSelectionViewController.h"
#import "SBNavigationBar.h"
#import "SBNavigationBarDetailView.h"
#import "SBAppDelegate.h"
#import "SBSelectableOptionTableViewCell.h"
#import "SBDefectSelectionViewController.h"
#import "SBReservationManager.h"
#import "SBColorManager.h"

@interface SBElementSelectionViewController ()

// fetches all objects that can be selected by the user from the CoreData store
- (void) retrieveSelectableObjects;

// indicates that the view should be reloaded and refresh its content
- (void) refreshContent;
- (void) showActivityView;

// the array which holds all element choices
@property (nonatomic, strong) NSArray *elements;

// the main managed object context of the application
@property (nonatomic, weak) NSManagedObjectContext *mainContext;

// the activity indicator that shows whether core data is accessed
@property (nonatomic, strong) UIActivityIndicatorView *activityIndicator;

@end

static NSString *SBElementSelectionCellReuseIdentifier  =   @"SBElementSelectionReuseIdentifier";

@implementation SBElementSelectionViewController

/*  The initializer method that gets called from the story board    */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    // call super initializer
    self = [super initWithCoder: aDecoder];

    if (self) {

        // set the main managed object context
        self.mainContext = [(SBAppDelegate *)[[UIApplication sharedApplication] delegate] managedObjectContext];
    }

    return self;
}

/*  When the view is loaded from memory this method is called and executes main initialization  */
- (void) viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    // set the title of the view controller
    [self setTitle: NSLocalizedString(@"location.viewcontroller.title", nil)];

    // set the nib used for reusing cells
    [self.elementTableView registerNib: [UINib nibWithNibName: @"SBSelectableOptionTableViewCell" bundle: [NSBundle
        mainBundle]] forCellReuseIdentifier: SBElementSelectionCellReuseIdentifier];

    // first reload the data
    [self retrieveSelectableObjects];
}

/*  when the view will appear on screen do final initialization */
- (void) viewWillAppear:(BOOL) animated
{
    // call super implementatiom
    [super viewWillAppear: animated];

    // create the detail view for the navigation bar
    SBNavigationBarDetailView *detailView = [[SBNavigationBarDetailView alloc] initWithDetailText: NSLocalizedString
        (@"element.viewcontroller.navigation.description", nil)];

    // configure the detail view
    [detailView setDetailTitle: NSLocalizedString(@"element.viewcontroller.navigation.detail.title", nil)];
    [detailView setDetailTintColor: [UIColor colorWithRed: 0.54296875 green: 0.8359375 blue: 0.73828125 alpha: 1.0]]
        ;
```

```
    [detailView setDetailImage: [UIImage imageNamed: @"OnderdeelStap"]];

    // add the detail view to the navigation bar
    [(SBNavigationBar *)self.navigationController.navigationBar setDetailView: detailView];
}

/*  when a memory warning is received try to free up memory */
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

/*  Prepare for the next view controller via this segue method  */
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // when the next view controller will be the defect selection
    if ([[segue identifier] isEqualToString: @"userDidSelectElement:"]) {

        // get the current selected indexpath and object that is selected
        NSIndexPath *selectedIndexPath = [self.elementTableView indexPathForSelectedRow];
        NSManagedObject *managedObject = [self.mainContext objectWithID: [self.elements objectAtIndex:
            [selectedIndexPath section]]];

        // commit the selected managed object and show next view controller
        [[SBReservationManager defaultReservationManager] processValue: managedObject
            forCurrentProceedingPropertyKey: @"element"];
        [[segue destinationViewController] setElementIdentifier: [managedObject objectID]];
    }
}

/*  Start to load the objects to show as selectable */
- (void) retrieveSelectableObjects
{
    // create the activity indicator to show that the data is being loaded
    [self showActivityView];

    // create a new managed object context to load the data from
    NSManagedObjectContext *managedObjectContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:
        NSPrivateQueueConcurrencyType];
    [managedObjectContext setPersistentStoreCoordinator: [(SBAppDelegate *)[[UIApplication sharedApplication]
        delegate] persistentStoreCoordinator]];

    // perform the block that loads the data
    [managedObjectContext performBlock: ^{

        // get the managed object that was selected in the previous view controller
        NSManagedObject *previousManagedObject = [self.mainContext objectWithID: self.spaceIdentifier];

        // create the fetch request for objects to retrieve
        NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] initWithEntityName: @"Element"];
        [fetchRequest setPredicate: [NSPredicate predicateWithFormat: @"space.spaceId = %@", [previousManagedObject
            valueForKeyPath: @"spaceId"]]];

        // create and add the sort descriptor
        NSSortDescriptor *sortDescriptor = [NSSortDescriptor sortDescriptorWithKey: @"elementDescription" ascending:
            YES];
        [fetchRequest setSortDescriptors: [NSArray arrayWithObject: sortDescriptor]];

        // create error object and lock the managed object context
        NSError *error = nil;
        [managedObjectContext lock];

        // execute the fetch request
        NSArray *results = [managedObjectContext executeFetchRequest: fetchRequest error: &error];

        // unlock the context
        [managedObjectContext unlock];

        // when no error occured set the rooms array
        if (error == nil) {

            // create a new array with all object ID's fetched in the background
            NSMutableArray *objectIDs = [NSMutableArray array];

            // add each object ID to the array
            for (NSManagedObject *managedObject in results) {
                [objectIDs addObject: [managedObject objectID]];
            }

            // set the new array and perform the refreshing of the view on the main thread
            self.elements = [NSArray arrayWithArray: objectIDs];
            [self performSelectorOnMainThread: @selector(refreshContent) withObject: nil waitUntilDone: NO];
        }
    }];
}
```

```
/*  Gets called on the main thread and indicates that new data is available to be shown */
- (void) refreshContent
{
    // set the tableview to not be hidden
    [self.elementTableView setHidden: NO];

    // remove the activity indicator when it is added to the view
    if ([self.activityIndicator superview] != nil) {

        // stop animation and remove from memory
        [self.activityIndicator stopAnimating];
        [self.activityIndicator removeFromSuperview];
        self.activityIndicator = nil;
    }

    // reload table data
    [self.elementTableView reloadData];
}

/*  When the data is being loaded from CoreData this method shows the activity indicator    */
- (void) showActivityView
{
    // hide the tableview
    [self.elementTableView setHidden: YES];

    // create and configure the activity indicator view
    self.activityIndicator = [[UIActivityIndicatorView alloc] initWithActivityIndicatorStyle:
        UIActivityIndicatorViewStyleGray];
    [self.activityIndicator setFrame: self.view.bounds];
    [self.activityIndicator setCenter: CGPointMake((self.view.frame.size.width / 2), (self.view.frame.size.height /
        2))];

    // add to the view and start the animation
    [self.view addSubview: self.activityIndicator];
    [self.activityIndicator startAnimating];
}

#pragma mark    -
#pragma mark    UITableView datasource & delegate

/*  Returns the amount of sections in the tableview, each section represents a selectable option    */
- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView
{
    // return the amount of objects in the element array
    return [self.elements count];
}

/*  Returns the amount of rows per section, in this case always one */
- (NSInteger) tableView:(UITableView *) tableView numberOfRowsInSection:(NSInteger) section
{
    return 1;
}

/*  Returns the configured cell for the given indexpath */
- (UITableViewCell *) tableView:(UITableView *) tableView cellForRowAtIndexPath:(NSIndexPath *) indexPath
{
    // dequeue a cell from the tableview, is always an object because the nib is registered in view did load
    SBSelectableOptionTableViewCell *cell = (SBSelectableOptionTableViewCell *)[tableView
        dequeueReusableCellWithIdentifier: SBElementSelectionCellReuseIdentifier];

    // get the corresponding object for this indexpath
    NSManagedObject *currentObject = (NSManagedObject *)[self.mainContext objectWithID: [self.elements
        objectAtIndex: [indexPath section]]];

    // set the text label of the cell
    [[cell optionLabel] setText: [[currentObject valueForKeyPath: @"elementDescription"] uppercaseString]];
    [cell setLayerTintColor: [[SBColorManager defaultColors] requestNavigationColor]];

    return cell;
}

/*  Returns the height of the header views in the tableview */
- (CGFloat) tableView:(UITableView *) tableView heightForHeaderInSection:(NSInteger) section
{
    // only the first section has a higher header
    if (section == 0) {
        return 20;
    } else {
        return 4;
    }
}

/*  Return the appended footer height for all cells */
- (CGFloat) tableView:(UITableView *) tableView heightForFooterInSection:(NSInteger) section
{
    // only the last cell has a higher footer for the white space
    if (section == ([self.elements count] - 1)) {
```

```
        return 10;
    } else {
        return 4;
    }
}

/*  Gets notified when the user has selected a table view cell, should show the next segue  */
- (void) tableView:(UITableView *) tableView didSelectRowAtIndexPath:(NSIndexPath *) indexPath
{
    // perform the next segue
    [self performSegueWithIdentifier: @"userDidSelectElement:" sender: self];
}

@end
```

```
//
//  SBHomeViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 16-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBViewController.h"
#import "SBReservationManagerDelegate.h"

@interface SBHomeViewController : SBViewController <UITableViewDelegate, UITableViewDataSource,
    SBReservationManagerDelegate>

// the outlet to the menu tableview with all default application options
@property (nonatomic, weak) IBOutlet UITableView *menuTableView;

@end
```

```
//
//  SBHomeViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 16-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBHomeViewController.h"
#import "SBHomeTableViewCell.h"
#import "SBNavigationBar.h"
#import "SBNavigationBarDetailView.h"
#import "SBUser.h"
#import "SBPersonalDataViewController.h"
#import "SBReservationManager.h"
#import "SBColorManager.h"

@interface SBHomeViewController ()

// the activity description text shown in the middle cell
@property (nonatomic, strong) NSString *activityDescription;

@end

// the reuse identifier used for home view controller cells
static NSString *SBHomeViewControllerCellReuseIdentifier    =   @"SBHomeViewControllerCellReuseIdentifier";

@implementation SBHomeViewController

- (void) viewDidLoad
{
    [super viewDidLoad];

    // Do any additional setup after loading the view.
    [self.navigationItem setTitle: NSLocalizedString(@"home.viewcontroller.title", nil)];

    // declare the specific class that is used for table view cells
    [self.menuTableView registerNib: [UINib nibWithNibName: @"SBHomeTableViewCell" bundle: nil]
        forCellReuseIdentifier:SBHomeViewControllerCellReuseIdentifier];

    // set the manager's activity delegate
    [[SBReservationManager defaultReservationManager] setActivityDelegate: self];
}

/*  When the view will appear we need to reset the detail navigation view   */
- (void) viewWillAppear:(BOOL)animated
{
    // the toolbar cannot be visible in this view
    [self.navigationController setToolbarHidden: YES animated: YES];

    SBNavigationBarDetailView *detail = [[SBNavigationBarDetailView alloc] initWithDetailText:
        NSLocalizedString(@"home.viewcontroller.navigation.description", nil)];
    [detail setDetailImage: [UIImage imageNamed: @"BouwMeesterLogo"]];

    // set the navigation bar detail view
    SBNavigationBar *navigationBar = (SBNavigationBar *)self.navigationController.navigationBar;
    [navigationBar setDetailView: detail];

    // when visible cancel any unfinished reservation
    [[SBReservationManager defaultReservationManager] didCancelReservation];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

/*  Perform a segue with the given segue object, prepare and set information for the next object */
- (void) prepareForSegue:(UIStoryboardSegue *) segue sender:(id) sender
{
    // when the personal data page will be shown
    if ([[segue identifier] isEqualToString: @"userWillEditPersonalData:"]) {

        // when the submit button is pressed on the view controller the location selector should be shown
        if (![[SBUser sharedUser] isValidated]) {
            [[segue destinationViewController] setShouldPopOnSubmit: NO];
        }
    }
}

#pragma mark    -
#pragma mark    UITableView datasource & delegate

/*  Returns the amount of sections in the tableview */
- (NSInteger) numberOfSectionsInTableView:(UITableView *) tableView
{
```

```
    // three sections of one cell
    return 3;
}

/*  Returns the amount of cells that will be shown for a secific section in the tableview   */
- (NSInteger) tableView:(UITableView *) tableView numberOfRowsInSection:(NSInteger) section
{
    // there is only one cell per section
    return 1;
}

/*  Returns the cell object for a specific indexpath in the tableview   */
- (UITableViewCell *) tableView:(UITableView *) tableView cellForRowAtIndexPath:(NSIndexPath *) indexPath
{
    // specify the reuse indentifier
    SBHomeTableViewCell *cell = (SBHomeTableViewCell *)[tableView dequeueReusableCellWithIdentifier:
        SBHomeViewControllerCellReuseIdentifier];

    // when no cell could be dequeued create a new cell
    if (cell == nil) {
        cell = [[SBHomeTableViewCell alloc] initWithStyle: UITableViewCellStyleDefault reuseIdentifier:
            SBHomeViewControllerCellReuseIdentifier];
    }

    // check indexpath to add the correct content to the correct cell
    if ([indexPath section] == 0) {

        // set the title to the corresponding localized string for adding a new request
        [[cell actionTextLabel] setText: NSLocalizedString(@"home.viewcontroller.tableview.create.title", nil)];
        [[cell actionImageView] setImage: [UIImage imageNamed: @"VerzoekIndienen"]];

    } else if ([indexPath section] == 1) {

        // set the title text to the localized string of all current requests
        [[cell actionTextLabel] setText: NSLocalizedString(@"home.viewcontroller.tableview.current.title", nil)];
        [[cell actionImageView] setImage: [UIImage imageNamed: @"LopendeVerzoeken"]];

        // when a activity description is set show it
        if (self.activityDescription != nil) {
            [cell setCellDescription: self.activityDescription];
            [cell setShouldIndicateActivity: YES];
        }

    } else if ([indexPath section] == 2) {

        // set the title of the label to a localized string of my info
        [[cell actionTextLabel] setText: NSLocalizedString(@"home.viewcontroller.tableview.info.title", nil)];
        [[cell actionImageView] setImage: [UIImage imageNamed: @"MijnGegevens"]];
    }

    // the cells do not have a selection style
    [cell setSelectionStyle: UITableViewCellSelectionStyleNone];

    return cell;
}

/*  Defines the height for each table view cell in the tableview    */
- (CGFloat) tableView:(UITableView *) tableView heightForRowAtIndexPath:(NSIndexPath *) indexPath
{
    // the height of an image is 50 so the cell is exactly that height
    return 50;
}

/*  Define the height of a header view in the tableview, set because the default space between sections is too high
    */
- (CGFloat) tableView:(UITableView *) tableView heightForHeaderInSection:(NSInteger) section
{
    // only for the first section the height should be higher than normal
    if (section == 0) {
        return 15;
    }

    return 5;
}

/*  Define the height of a footer view in the tableview, set because the default space between sections is too high
    */
- (CGFloat) tableView:(UITableView *) tableView heightForFooterInSection:(NSInteger) section
{
    return 5;
}

/*  When the user pressed a table view cell this method is called   */
- (void) tableView:(UITableView *) tableView didSelectRowAtIndexPath:(NSIndexPath *) indexPath
{
    // switch to the correct action to undertake
    switch ([indexPath section]) {
```

```
        case 0:

            // first let the manager create a new reservation
            [[SBReservationManager defaultReservationManager] willGenerateReservation];
            [[SBReservationManager defaultReservationManager] setCurrentProceedingIndex: 0];

            // check whether the user has already provided valid user credentials, when true the new request can be
                started immediatly, otherwise the  credentials should be asked first
            if ([[SBUser sharedUser] isValidated]) {
                [self performSegueWithIdentifier: @"userAlreadyHasValidCredentials:" sender: self];
            } else {
                [self performSegueWithIdentifier: @"userWillEditPersonalData:" sender: self];
            }

            break;

        case 1:
            // create the overview of all maintenance requests
            [self performSegueWithIdentifier: @"userDidPressAllReservationCell:" sender: self];
            break;

        case 2:

            // start the segue for the personal data page
            [self performSegueWithIdentifier: @"userWillEditPersonalData:" sender: self];
            break;

        default:
            // do nothing on default
            break;
    }
}

#pragma mark    -
#pragma mark    SBReservationManager delegate

/*  Gets called when the description text has changed of the activity in the reservation manager    */
- (void) manager:(SBReservationManager *) manager didUpdateActivityWithDescription:(NSString *) description
{
    // set the current activity description
    self.activityDescription = description;

    // reload the cell
    [self.menuTableView reloadSections: [NSIndexSet indexSetWithIndex: 1] withRowAnimation:
        UITableViewRowAnimationAutomatic];
}

/*  Gets called when all activity has ended */
- (void) managerDidEndActivity:(SBReservationManager *) manager
{
    // reser the description and reload
    self.activityDescription = nil;
    [self.menuTableView reloadSections: [NSIndexSet indexSetWithIndex: 1] withRowAnimation:
        UITableViewRowAnimationAutomatic];
}

@end
```

```
//
//  SBLocationSelectionViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 21-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBViewController.h"

@interface SBLocationSelectionViewController : SBViewController <UICollectionViewDataSource,
    UICollectionViewDelegate, UICollectionViewDelegateFlowLayout>

// the collection view holding the tifferent choices
@property (nonatomic, weak) IBOutlet UICollectionView *locationCollectionView;

@end
```

```
//
//  SBLocationSelectionViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 21-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBLocationSelectionViewController.h"
#import "SBNavigationBar.h"
#import "SBNavigationBarDetailView.h"
#import "SBCollectionViewLocationCell.h"
#import "SBRoomSelectionViewController.h"
#import "SBAppDelegate.h"
#import "SBReservationManager.h"

@interface SBLocationSelectionViewController ()

// processes the fetched object ID on the main thread
- (void) processLocationObjectID:(NSManagedObjectID *) objectID;

@end

// identifier for reusable cells
static NSString *SBLocationSelectionCellReuseIdentifier  =   @"SBLocationSelectionReuseIdentifier";

@implementation SBLocationSelectionViewController

/*  Initializer from a storyboard object    */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    // call super implementation
    self = [super initWithCoder: aDecoder];

    if (self) {

    }
    return self;
}

/*  When the view is loaded this method gets called for further initialization  */
- (void) viewDidLoad
{
    // call super implementation
    [super viewDidLoad];

    // set the title and prepare the new detail view
    [self setTitle: NSLocalizedString(@"location.viewcontroller.title", nil)];

    // register the correct class to use for this cell
    [self.locationCollectionView registerNib: [UINib nibWithNibName: @"SBCollectionViewLocationCell" bundle:
        [NSBundle mainBundle]] forCellWithReuseIdentifier: SBLocationSelectionCellReuseIdentifier];

    // create the flow layout for the collection view
    UICollectionViewFlowLayout *collectionViewFlowLayout = [[UICollectionViewFlowLayout alloc] init];
    [collectionViewFlowLayout setItemSize: CGSizeMake(130, 130)];
    [collectionViewFlowLayout setScrollDirection: UICollectionViewScrollDirectionVertical];

    // set the flow layout
    [self.locationCollectionView setCollectionViewLayout: collectionViewFlowLayout];

    // hide toolbar when visible
    [self.navigationController setToolbarHidden: YES animated: YES];
}

/*  When the view will appear on screen method gets called append the detail view   */
- (void) viewWillAppear:(BOOL) animated
{
    // call super implementatiom
    [super viewWillAppear: animated];

    // create the new detail view
    SBNavigationBarDetailView *detailView = [[SBNavigationBarDetailView alloc] initWithDetailText: NSLocalizedString
        (@"location.viewcontroller.detail.description", nil)];

    // configure the detail view with a title, tint color and image
    [detailView setDetailTitle: NSLocalizedString(@"location.viewcontroller.detail.title", nil)];
    [detailView setDetailTintColor: [UIColor colorWithRed: 0.54296875 green: 0.8359375 blue: 0.73828125 alpha: 1.0]]
        ;
    [detailView setDetailImage: [UIImage imageNamed: @"LocatieStapAfbeelding"]];

    // set the new detail view to the navigation bar
    [(SBNavigationBar *)self.navigationController.navigationBar setDetailView: detailView];
}

/*  Gets called when a memory warning is received   */
- (void) didReceiveMemoryWarning
```

```
{
    // call super implementation
    [super didReceiveMemoryWarning];
}

/*  Prepare for the next segue  */
- (void) prepareForSegue:(UIStoryboardSegue *) segue sender:(id) sender
{
    // when the next segue is the room view controller append data
    if ([[segue identifier] isEqualToString: @"userDidSelectLocation:"]) {

        // get the selected indexpath and set the location identifier for the destination view controller
        NSIndexPath *selectedItem = [[self.locationCollectionView indexPathsForSelectedItems] objectAtIndex: 0];
        __block NSNumber *locationIdentifier = nil;

        // switch to set the correct location identifier
        switch (selectedItem.row) {

            case 0:

                // outside the house is 2
                locationIdentifier = [NSNumber numberWithInteger: 2];
                break;
            case 1:

                // inside the house is 3
                locationIdentifier = [NSNumber numberWithInteger: 3];
                break;
            case 2:

                // around the house is 4
                locationIdentifier = [NSNumber numberWithInteger: 4];
                break;
            case 3:

                // general locations is 1
                locationIdentifier = [NSNumber numberWithInteger: 1];
                break;
        }

        // create a new managed object context to fetch the object
        NSManagedObjectContext *managedObjectContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:
            NSPrivateQueueConcurrencyType];
        [managedObjectContext setPersistentStoreCoordinator: [(SBAppDelegate *)[[UIApplication sharedApplication]
            delegate] persistentStoreCoordinator]];

        // perform the block that fetches the location in the background
        [managedObjectContext performBlock: ^{

            // create the fetch request for the location
            NSFetchRequest *locationFetchRequest = [[NSFetchRequest alloc] initWithEntityName: @"Location"];
            [locationFetchRequest setPredicate: [NSPredicate predicateWithFormat: @"locationId = %@",
                locationIdentifier]];

            // execute the fetch request
            NSError *error = nil;
            NSArray *locations = [managedObjectContext executeFetchRequest: locationFetchRequest error: &error];

            // when the error is still nil no errors have occured
            if (error == nil && [locations count] == 1) {

                // perform the actual setting of the location object on the main thread
                [self performSelectorOnMainThread: @selector(processLocationObjectID:) withObject: [[locations
                    objectAtIndex: 0] objectID] waitUntilDone: NO];
            }
        }];

        // set the calculated location identifier for the next view controller
        [[segue destinationViewController] setLocationIdentifier: locationIdentifier];
    }
}

/*  Gets called from the background and sets the proceedings location identifier object */
- (void) processLocationObjectID:(NSManagedObjectID *) objectID
{
    // get the main object context and fetch the object corresponding to the given object ID
    NSManagedObjectContext *mainContext = [(SBAppDelegate *)[[UIApplication sharedApplication] delegate]
        managedObjectContext];
    NSManagedObject *fetchedLocation = [mainContext objectWithID: objectID];

    // process the object in the reservation manager
    [[SBReservationManager defaultReservationManager] processValue: fetchedLocation forCurrentProceedingPropertyKey:
        @"location"];
}

#pragma mark    -
#pragma mark    UICollectionView delegate & datasource
```

```
/*  In the collection view only one section is present  */
- (NSInteger) numberOfSectionsInCollectionView:(UICollectionView *) collectionView
{
    return 1;
}

/*  Returns the amount of items in the section provided */
- (NSInteger) collectionView:(UICollectionView *) collectionView numberOfItemsInSection:(NSInteger) section
{
    return 4;
}

/*  Returns the appropriate collection view cell for the passed indexPath   */
- (UICollectionViewCell *) collectionView:(UICollectionView *) collectionView cellForItemAtIndexPath:(NSIndexPath *)
    indexPath
{
    // dequeue a cell from the provided collection view
    SBCollectionViewLocationCell *cell = (SBCollectionViewLocationCell *)[collectionView
        dequeueReusableCellWithReuseIdentifier: SBLocationSelectionCellReuseIdentifier forIndexPath: indexPath];
    [cell setBackgroundColor: [UIColor blackColor]];

    // switch to fill the correct cell information
    switch ([indexPath row]) {

        case 0:
            // set the background image and texts of the cell for outside of the house
            [[cell imageView] setImage: [UIImage imageNamed: @"BuitenkantWoning"]];
            [[cell titleLabel] setText: NSLocalizedString(@"location.viewcontroller.outside.cell.title", nil)];
            [[cell footnoteLabel] setText: NSLocalizedString(@"location.viewcontroller.outside.cell.footnote", nil)]
                ;
            break;

        case 1:

            // prepare the cell for the inside cell description
            [[cell imageView] setImage: [UIImage imageNamed: @"InDeWoning"]];
            [[cell titleLabel] setText: NSLocalizedString(@"location.viewcontroller.inside.cell.title", nil)];
            [[cell footnoteLabel] setText: NSLocalizedString(@"location.viewcontroller.inside.cell.footnote", nil)];
            break;

        case 2:

            // prepare the cell for around the house description
            [[cell imageView] setImage: [UIImage imageNamed: @"RondomDeWoning"]];
            [[cell titleLabel] setText: NSLocalizedString(@"location.viewcontroller.around.cell.title", nil)];
            [[cell footnoteLabel] setText: NSLocalizedString(@"location.viewcontroller.around.cell.footnote", nil)];
            break;

        case 3:

            // prepare the cell for general spaces description
            [[cell imageView] setImage: [UIImage imageNamed: @"AlgemeneRuimten"]];
            [[cell titleLabel] setText: NSLocalizedString(@"location.viewcontroller.general.cell.title", nil)];
            [[cell footnoteLabel] setText: NSLocalizedString(@"location.viewcontroller.general.cell.footnote", nil)]
                ;
            break;
    }

    // set the border radius of each cell
    cell.layer.masksToBounds = YES;
    cell.layer.cornerRadius = 3.0f;
    return cell;
}

/*  when a cell is selected in the collection view this method is called    */
- (void) collectionView:(UICollectionView *) collectionView didSelectItemAtIndexPath:(NSIndexPath *) indexPath
{
    // start the next segue for room selection
    [self performSegueWithIdentifier: @"userDidSelectLocation:" sender: self];
}

/*  Returns the top edge insets used for each section, in this collection view we only have one section */
- (UIEdgeInsets) collectionView:(UICollectionView *)collectionView layout:(UICollectionViewLayout *)
    collectionViewLayout insetForSectionAtIndex:(NSInteger)section
{
    // keep 20 points clear of every border
    return UIEdgeInsetsMake(20, 20, 20, 20);
}

/*  Returns the minimum space used as white space between collection view cells  */
- (CGFloat) collectionView:(UICollectionView *)collectionView layout:(UICollectionViewLayout *)collectionViewLayout
    minimumInteritemSpacingForSectionAtIndex:(NSInteger)section
{
    // keep mimimum white space of 20 points
    return 20;
}
```

```
/*  Returns the vertical spacing between collection view cells  */
- (CGFloat) collectionView:(UICollectionView *)collectionView layout:(UICollectionViewLayout *)collectionViewLayout
    minimumLineSpacingForSectionAtIndex:(NSInteger)section
{
    // also vertical spacing of 20 points
    return 20;
}

@end
```

```
//
//  SBOptionalInformationViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 22-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBViewController.h"

@class SBDefectImageDescriptor;

@interface SBOptionalInformationViewController : SBViewController <UITextViewDelegate, UIActionSheetDelegate,
    UINavigationControllerDelegate, UIImagePickerControllerDelegate>

// the contents are positioned in a scrollview
@property (nonatomic, weak) IBOutlet UIScrollView *contentScrollView;

// the textview which the user can edit
@property (nonatomic, strong) UITextView *contentTextView;

// the image descriptor class that represents the image and processing
@property (nonatomic, strong) SBDefectImageDescriptor *imageDescriptor;

@end
```

```
//
//  SBOptionalInformationViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 22-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBOptionalInformationViewController.h"
#import "SBNavigationBar.h"
#import "SBNavigationBarDetailView.h"
#import "SBDefectImageDescriptor.h"
#import "SBReservationManager.h"
#import "SBColorManager.h"

// for media types
#import <MobileCoreServices/MobileCoreServices.h>

@interface SBOptionalInformationViewController ()

// when the camera button was pressed this method gets called and takes actions
- (IBAction) cameraBarButtonItemPressed:(UIBarButtonItem *) sender;

// when the user has pressed the confirm button
- (void) confirmButtonPressed:(UIBarButtonItem *) sender;

// generates the attributed string for the Label
- (NSAttributedString *) attributedTextForNavigationTitle;

// when the keyboard will show or hide
- (void) keyboarDidShow:(NSNotification *) notification;
- (void) keyboardWillHide:(NSNotification *) notification;

// when the user pressed the background view the textview should resign first responder when needed
- (void) userDidTapViewBackground:(id) sender;

// draws the processed image on screen and saves the content to the reservation manager
- (void) refreshContent;
- (void) saveContent;

// the font that is used to draw the text in the textview
@property (nonatomic, strong) UIFont *textViewFont;

// the view container for the text and image
@property (nonatomic, strong) UIView *imageAndTextContainer;

// whether the placeholder text is being showed
@property (nonatomic) BOOL isShowingPlaceholderText;
@property (nonatomic) BOOL isObservingImageProcessing;

// specifies whether the user has given an input value, image or text
@property (nonatomic) BOOL hasContent;

// the right bar button item for the confirm title
@property (nonatomic, strong) UIButton *buttonView;

@end

@implementation SBOptionalInformationViewController

/*  Initializer method which gets called from the storyboard instance   */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    // call super initializer
    self = [super initWithCoder: aDecoder];

    if (self) {

        // set the used textview font
        self.textViewFont = [UIFont systemFontOfSize: 14.0f];
        self.isObservingImageProcessing = NO;
        self.hasContent = NO;
    }

    return self;
}

/*  when deallocated remove from notification center    */
- (void) dealloc
{
    [[NSNotificationCenter defaultCenter] removeObserver: self];
}

/*  When the view is loaded into memory this method is called and allows for interface initialization   */
- (void) viewDidLoad
{
    [super viewDidLoad];
```

```
// set the title of the view controller
[self setTitle: NSLocalizedString(@"location.viewcontroller.title", nil)];

// set the toolbar visible
[self.navigationController setToolbarHidden: NO animated: YES];

// create the toolbar for the textview keyboard with the default toolbar dimensions
UIToolbar *textViewToolbar = [[UIToolbar alloc] initWithFrame: self.navigationController.toolbar.frame];

// the button that will hide the keyboard
UIBarButtonItem *hideButton = [[UIBarButtonItem alloc] initWithTitle:
    NSLocalizedString(@"optional.viewcontroller.textview.toolbar.hide.title", nil) style:
    UIBarButtonItemStylePlain target: self action: @selector(userDidTapViewBackground:)];
UIBarButtonItem *flexibleSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
    UIBarButtonSystemItemFlexibleSpace target: nil action: nil];

// set the toolbar items
[textViewToolbar setItems: [NSArray arrayWithObjects: flexibleSpace, hideButton, nil]];

// fetch the previous image when available and the inout text
NSString *optionalText = (NSString *)[[SBReservationManager defaultReservationManager]
    valueForCurrentProceedingProperty: @"proceedingDescription"];
self.imageDescriptor = (SBDefectImageDescriptor *)[[SBReservationManager defaultReservationManager]
    valueForCurrentProceedingProperty: @"imageDescriptor"];

// create the textview for the text content
self.contentTextView = [[UITextView alloc] initWithFrame: CGRectMake(10, 10, 280, 132)];
[self.contentTextView setInputAccessoryView: textViewToolbar];
[self.contentTextView setDelegate: self];

// set the previous added text when available
if (optionalText != nil && [optionalText length] > 0) {
    [self.contentTextView setText: optionalText];
}

// set the placeholder text
if ([[self.contentTextView text] length] == 0) {

    self.isShowingPlaceholderText = YES;
    [self.contentTextView setText: NSLocalizedString(@"optional.viewcontroller.textview.placeholder", nil)];
    self.isShowingPlaceholderText = YES;

    // set the placeholder font
    [self.contentTextView setFont: [UIFont italicSystemFontOfSize: 11]];
    [self.contentTextView setTextColor: [UIColor lightGrayColor]];

} else {

    // set the text font
    [self.contentTextView setFont: self.textViewFont];
}

// create the text and image container view
self.imageAndTextContainer = [[UIView alloc] initWithFrame: CGRectZero];
[self.imageAndTextContainer setBackgroundColor: [UIColor whiteColor]];

// make the border mask and radius
[[self.imageAndTextContainer layer] setMasksToBounds: YES];
[[self.imageAndTextContainer layer] setCornerRadius: 3.0f];

// add the container to the content scrollview
[self.imageAndTextContainer addSubview: self.contentTextView];
[self.contentScrollView addSubview: self.imageAndTextContainer];

// create the toolbar items
UIBarButtonItem *imageBarButtonItem = [[UIBarButtonItem alloc] initWithImage: [UIImage imageNamed:
    @"CameraButton"] style: UIBarButtonItemStylePlain target: self action: @selector(cameraBarButtonItemPressed:
    )];
[imageBarButtonItem setTintColor: [[SBColorManager defaultColors] requestNavigationColor]];

// the button which has to be pressed to go to the next page
self.buttonView = [[UIButton alloc] initWithFrame: CGRectMake(180, 0, 140, 44)];
[self.buttonView addTarget: self action: @selector(confirmButtonPressed:) forControlEvents:
    UIControlEventTouchUpInside];
[self.buttonView setBackgroundColor: [UIColor colorWithRed: 0.54296875 green: 0.8359375 blue: 0.73828125 alpha:
    1.0]];
[[self.buttonView titleLabel] setFont: [UIFont systemFontOfSize: 13.0f]];

// set the title of the button
if (self.hasContent) {
    [self.buttonView setTitle: NSLocalizedString(@"optional.viewcontroller.toolbar.next.title", nil) forState:
        UIControlStateNormal];
} else {
    [self.buttonView setTitle: NSLocalizedString(@"optional.viewcontroller.toolbar.skip.title", nil) forState:
        UIControlStateNormal];
}
```

```
    // create the bar button item
    UIBarButtonItem *doneButtonItem = [[UIBarButtonItem alloc] initWithCustomView: self.buttonView];

    // the fixed space makes sure the right bar button item is placed against the right border of the toolbar
    UIBarButtonItem *fixedSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
        UIBarButtonSystemItemFixedSpace target: nil action: nil];
    fixedSpace.width -= 16.0f;

    // set the new toolbar items
    [self setToolbarItems: [self.toolbarItems arrayByAddingObjectsFromArray: [NSArray arrayWithObjects:
        doneButtonItem, fixedSpace, nil]] animated: YES];

    // get notified about keyboard notifications
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboarDidShow:) name:
        UIKeyboardDidShowNotification object: nil];
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboardWillHide:) name:
        UIKeyboardWillHideNotification object: nil];

    // create the tap gesture recognizer for the background view, when pressed the keyboard should beb hidden when
        shown
    UITapGestureRecognizer *tapGesture = [[UITapGestureRecognizer alloc] initWithTarget: self action: @selector
        (userDidTapViewBackground:)];
    [self.view addGestureRecognizer: tapGesture];
}

/*  Gets called when a memory warning is received, clean up resources   */
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

/*  when the view will appear on screen to last initialization like toolbar showing */
- (void) viewWillAppear:(BOOL) animated
{
    // call super implementation
    [super viewWillAppear: animated];

    // set the default height of the textview
    CGFloat textHeight = 152;

    // when the height is larger than this default value change it
    if (self.contentTextView.frame.size.height > textHeight) {
        textHeight = (self.contentTextView.frame.size.height + 20);
    }

    // set the frame of the text and image container
    [self.imageAndTextContainer setFrame: CGRectMake(10, 20, 300, textHeight)];

    // set the frame and add to screen
    [self.contentScrollView setContentSize: CGSizeMake(self.view.frame.size.width, (textHeight + 40))];

    // create the new detail view
    SBNavigationBarDetailView *detailView = [[SBNavigationBarDetailView alloc] initWithDetailText: NSLocalizedString
        (@"optional.viewcontroller.navigation.description", nil)];

    // further configure the detail view
    [detailView setAttributedTitleString: [self attributedTextForNavigationTitle]];
    [detailView setDetailTintColor: [UIColor colorWithRed: 0.54296875 green: 0.8359375 blue: 0.73828125 alpha: 1.0]]
        ;
    [detailView setDetailImage: [UIImage imageNamed: @"OptioneelStap"]];

    // append the detail view to the navigation bar
    [(SBNavigationBar *)self.navigationController.navigationBar setDetailView: detailView];

    // when the image descriptor is set and this view is not observing add observer object
    if (self.imageDescriptor != nil && !self.isObservingImageProcessing) {
        [self.imageDescriptor addObserver: self forKeyPath: @"isProcessing" options: NSKeyValueObservingOptionNew
            context: NULL];
        self.isObservingImageProcessing = YES;
    }
}

/*  When the view will disappear remove key value observer  */
- (void) viewWillDisappear:(BOOL) animated
{
    // call super implemantetion
    [super viewWillDisappear: animated];

    // remove observer when needed
    if (self.isObservingImageProcessing) {
        [self.imageDescriptor removeObserver: self forKeyPath: @"isProcessing"];
        self.isObservingImageProcessing = NO;
    }
}
```

```
/*  Generates the attributed text that will be used in the navigation detail title  */
- (NSAttributedString *) attributedTextForNavigationTitle
{
    // the font for the normal part of the string and the optional italic font
    UIFont *titleFont = [UIFont boldSystemFontOfSize: 13];
    UIFont *italicFont = [UIFont italicSystemFontOfSize: 11];

    // create the string attributes for the normal part of the string
    NSDictionary *normalAttributes = @{NSFontAttributeName: titleFont};
    NSDictionary *italicAttributes = @{NSFontAttributeName: italicFont};

    // get the strings to be inserted in the attributed text
    NSString *normalString = NSLocalizedString(@"optional.viewcontroller.navigation.detail.title", nil);
    NSString *italicString = NSLocalizedString(@"optional.viewcontroller.navigation.detail.title.addition", nil);

    // create the attributed string with the full string that will be shown, set the italic part of the string
    NSMutableAttributedString *attributedString = [[NSMutableAttributedString alloc] initWithString: [NSString
        stringWithFormat: @"%@ %@", normalString, italicString] attributes: normalAttributes];
    [attributedString setAttributes: italicAttributes range: NSMakeRange(normalString.length + 1, italicString.
        length)];

    // return the generated string
    return attributedString;
}

/*  When the camera button was pressed show the options from this method    */
- (IBAction) cameraBarButtonItemPressed:(UIBarButtonItem *) sender
{
    // check whether a choice should be made between selecting or creating a new image
    if ([UIImagePickerController isSourceTypeAvailable: UIImagePickerControllerSourceTypeCamera]) {

        // create the actionsheet and set the title
        UIActionSheet *actionSheet = [[UIActionSheet alloc] initWithTitle:
            NSLocalizedString(@"optional.viewcontroller.actionsheet.title", nil) delegate: self cancelButtonTitle:
            NSLocalizedString(@"optional.viewcontroller.actionsheet.cancel.title", nil) destructiveButtonTitle: nil
            otherButtonTitles: NSLocalizedString(@"optional.viewcontroller.actionsheet.new.title", nil),
            NSLocalizedString(@"optional.viewcontroller.actionsheet.existing.title", nil), nil];

        // show the actionsheet in the current view
        [actionSheet showFromToolbar: self.navigationController.toolbar];

    } else if ([UIImagePickerController isSourceTypeAvailable: UIImagePickerControllerSourceTypeSavedPhotosAlbum]) {

        // only the saved photos album is available, create the picker controller for the saved photos album
        UIImagePickerController *imagePickerController = [[UIImagePickerController alloc] init];
        [imagePickerController setSourceType: UIImagePickerControllerSourceTypeSavedPhotosAlbum];

        // set delegate and the only accepted media type: image
        [imagePickerController setDelegate: self];
        [imagePickerController setMediaTypes: [NSArray arrayWithObject: (NSString *)kUTTypeImage]];

        // present the image picker controller
        [self presentViewController: imagePickerController animated: YES completion: ^{

            // set that the navigation bar should not update its detail view
            [(SBNavigationBar *)self.navigationController.navigationBar setShouldAcceptNextDetailView: NO];
        }];
    } else {

        // no choices are available, this could be an error show alertview
        UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
            NSLocalizedString(@"optional.viewcontroller.sources.unavailable.title", nil) message:
            NSLocalizedString(@"optional.viewcontroller.sources.unavailable.description", nil) delegate: nil
            cancelButtonTitle: nil otherButtonTitles:
            NSLocalizedString(@"optional.viewcontroller.sources.unavailable.dismiss.title", nil), nil];

        // show the alert
        [alertView show];
    }
}

/*  When the confirm button was pressed by the user */
- (void) confirmButtonPressed:(UIBarButtonItem *) sender
{
    // activate the next segue
    [self performSegueWithIdentifier: @"userDidFinishOptionalOptions:" sender: self];
    [self saveContent];
}

/*  Saves the input content of the view to the reservation manager  */
- (void) saveContent
{
    // commit values to the reservation manager
    SBReservationManager *reservationManager = [SBReservationManager defaultReservationManager];

    // save text when set
    if ([[self.contentTextView text] length] > 0 && !self.isShowingPlaceholderText) {
```

```
        [reservationManager processValue: [self.contentTextView text] forCurrentProceedingPropertyKey:
            @"proceedingDescription"];
    }

    // when an image is provided save the image descriptor
    if (self.imageDescriptor != nil) {
        [reservationManager processValue: self.imageDescriptor forCurrentProceedingPropertyKey: @"imageDescriptor"];
    }
}

/*  Gets called when the user taps the background view, should resign first responder   */
- (void) userDidTapViewBackground:(UITapGestureRecognizer *) tapGesture
{
    // when the textview is first responder resign it
    if ([self.contentTextView isFirstResponder]) {
        [self.contentTextView resignFirstResponder];
    }
}

/*  Refresh the content on the screen   */
- (void) refreshContent
{
    // make the first button in the toolbar the photo selection button
    UIBarButtonItem *imageBarButtonItem = [[UIBarButtonItem alloc] initWithImage: [UIImage imageNamed:
        @"CameraButton"] style: UIBarButtonItemStylePlain target: self action: @selector(cameraBarButtonItemPressed:
        )];
    [imageBarButtonItem setTintColor: [UIColor colorWithRed: 0.54296875 green: 0.8359375 blue: 0.73828125 alpha: 1.0
        ]];

    // set the toolbar items
    NSArray *currentToolbarItems = [self.navigationController.toolbar items];

    // last item is for the horizontal spacing for last object
    UIBarButtonItem *fixedSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
        UIBarButtonSystemItemFixedSpace target: nil action: nil];
    fixedSpace.width -= 16.0f;
    [self setToolbarItems: [NSArray arrayWithObjects: imageBarButtonItem, [currentToolbarItems objectAtIndex: 1],
        [currentToolbarItems objectAtIndex: 2], fixedSpace, nil] animated: YES];

    // remove current image views from its container
    for (UIView *subview in [self.imageAndTextContainer subviews]) {

        // when an imageView is found remove it from superview
        if ([subview isKindOfClass: [UIImageView class]]) {
            [subview removeFromSuperview];
        }
    }

    // add the image in an imageview to the view container
    UIImageView *imageView = [[UIImageView alloc] initWithImage: [self.imageDescriptor imageThumbnail]];

    // calculate an aspect fit size for the image in the view container
    CGSize imageSize = [[self.imageDescriptor imageThumbnail] size];

    // when the image size width is bigger than the frame allows calculate the aspect fit height for the image view
    if (imageSize.width > self.imageAndTextContainer.frame.size.width) {

        // get the total amount of points that should be removed for a horizontal aspect fit
        CGFloat onePointPercent = 100 / imageSize.width;
        CGFloat totalPercentage = (onePointPercent * self.imageAndTextContainer.frame.size.width) / 100;

        // set the new frame of the image view and make the image aspect fit
        [imageView setFrame: CGRectMake(0, (self.contentTextView.frame.size.height + 20), self.imageAndTextContainer
            .frame.size.width, (imageSize.height * totalPercentage))];
        [imageView setContentMode: UIViewContentModeScaleAspectFit];

    } else {

        // make sure the image view is centered
        [imageView setContentMode: UIViewContentModeCenter];
        [imageView setFrame: CGRectMake(0, (self.contentTextView.frame.size.height + 20), self.imageAndTextContainer
            .frame.size.width, imageView.image.size.height)];
    }

    // add the image view to the view container
    [self.imageAndTextContainer setFrame: CGRectMake(self.imageAndTextContainer.frame.origin.x, self.
        imageAndTextContainer.frame.origin.y, self.imageAndTextContainer.frame.size.width, (self.contentTextView.
        frame.size.height + imageView.frame.size.height + 40))];
    [self.imageAndTextContainer addSubview: imageView];

    // append the content size of the scrollview
    [self.contentScrollView setContentSize: CGSizeMake(320, self.imageAndTextContainer.frame.size.height + 40)];
}

#pragma mark    -
#pragma mark    UIImagePickerController delegate
```

```
/*  When the user has selected a media image using the picker controller this method gets called with information
    */
- (void) imagePickerController:(UIImagePickerController *) picker didFinishPickingMediaWithInfo:(NSDictionary *)
    info
{
    // check for an already existing image descriptori
    if (self.imageDescriptor != nil) {

        // remove the image descriptor and backing file image
        [self.imageDescriptor deleteCurrentImage];

        // remove the observer and release the object
        if (self.isObservingImageProcessing) {
            [self.imageDescriptor removeObserver: self forKeyPath: @"isProcessing"];
            self.isObservingImageProcessing = NO;
        }

        // reset the image descriptor
        self.imageDescriptor = nil;
    }

    // when the metadata key is set, a new image is taken with the camera
    if ([info objectForKey: UIImagePickerControllerMediaMetadata]) {

        // save the image to disk when possible
        self.imageDescriptor = [[SBDefectImageDescriptor alloc] initWithImage: [info valueForKey:
            UIImagePickerControllerOriginalImage] metadata: [info valueForKey: UIImagePickerControllerMediaMetadata]
            ];

    } else if ([info objectForKey: UIImagePickerControllerReferenceURL]) {

        // an image is picked with from the camera roll / photo library
        self.imageDescriptor = [[SBDefectImageDescriptor alloc] initWithReferenceURL: [info valueForKey:
            UIImagePickerControllerReferenceURL] metaData: nil];

    } else {

        // no image was found in the dictionary, show error alert with appropriate information
        UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
            NSLocalizedString(@"optional.viewcontroller.result.unavailable.title", nil) message:
            NSLocalizedString(@"optional.viewcontroller.result.unavailable.description", nil) delegate: nil
            cancelButtonTitle: nil otherButtonTitles:
            NSLocalizedString(@"optional.viewcontroller.sources.unavailable.dismiss.title", nil), nil];

        // show the alertview
        [alertView show];
    }

    // when the image descriptor is successfully created start the image processing and show an image activity
        indicator
    if (self.imageDescriptor != nil) {

        // observer the isProcessing value of the imagedescriptor
        if (!self.isObservingImageProcessing) {

            // add key value listener
            [self.imageDescriptor addObserver: self forKeyPath: @"isProcessing" options:
                NSKeyValueObservingOptionNew context: NULL];
            self.isObservingImageProcessing = YES;
        }

        // start processing the image
        [self.imageDescriptor startProcessing];

        // replace the image choosing button with a loading indicator when the image is being processed
        UIActivityIndicatorView *buttonActivityIndicator = [[UIActivityIndicatorView alloc]
            initWithActivityIndicatorStyle: UIActivityIndicatorViewStyleGray];
        [buttonActivityIndicator startAnimating];
        UIBarButtonItem *activityBarbuttonItem = [[UIBarButtonItem alloc] initWithCustomView:
            buttonActivityIndicator];

        // get the current items of the toolbar
        NSArray *currentToolbarItems = [self.navigationController.toolbar items];
        [self setToolbarItems: [NSArray arrayWithObjects: activityBarbuttonItem, [currentToolbarItems objectAtIndex:
            1], [currentToolbarItems objectAtIndex: 2], nil] animated: YES];

        // set new button title
        [self.buttonView setTitle: NSLocalizedString(@"optional.viewcontroller.toolbar.next.title", nil) forState:
            UIControlStateNormal];
    }

    // dismiss the presented picker controller
    [self dismissViewControllerAnimated: YES completion: nil];
}

/*  Gets called when the image picker controller got cancelled by the user  */
- (void) imagePickerControllerDidCancel:(UIImagePickerController *) picker
```

```
{
    // dismiss the presented picker controller
    [self dismissViewControllerAnimated: YES completion: nil];
}


#pragma mark    -
#pragma mark    Key-Value observing

/*  When an object being observed by this object and a keypath changes its value this method gets called, we want to
    listen to the "isProcessing" keyPath   */
- (void) observeValueForKeyPath:(NSString *) keyPath ofObject:(id) object change:(NSDictionary *) change context:
    (void *) context
{
    // when the keyPath is "isProcessing" continue
    if ([keyPath isEqualToString: @"isProcessing"]) {

        // when processing is NO the image has been processed
        if (![object isProcessing])
        {
            // when the image descriptor encountered an error show the error view
            if ([self.imageDescriptor processingError] != nil) {

                // an error occured during the processing
                UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
                    NSLocalizedString(@"optional.viewcontroller.processing.failed.title", nil) message: [[self.
                    imageDescriptor processingError] localizedDescription] delegate: nil cancelButtonTitle: nil
                    otherButtonTitles:
                    NSLocalizedString(@"optional.viewcontroller.sources.unavailable.dismiss.title", nil), nil];

                // show the alertview, call from the main thread
                [alertView performSelectorOnMainThread: @selector(show) withObject: nil waitUntilDone: NO];

            } else {

                // show the processed image on screen by calling the refresh content method, this must be done on
                    the main thread
                [self performSelectorOnMainThread: @selector(refreshContent) withObject: nil waitUntilDone: NO];
            }

            // stop observing keypath when not already
            if (self.isObservingImageProcessing) {
                [object removeObserver: self forKeyPath: keyPath];
                self.isObservingImageProcessing = NO;
            }
        }
    }
}

#pragma mark    UIKeyboard methods

/*  When the keyboard will become available on the screen prepare the interface */
- (void) keyboarDidShow:(NSNotification *) notification
{
    // get the height of the keyboard frame
    CGFloat keyboardHeight = [[[notification userInfo] objectForKey: UIKeyboardFrameEndUserInfoKey] CGRectValue].
        size.height;

    // append the height to the scrollview
    [self.contentScrollView setFrame: CGRectMake(0, 0, self.view.frame.size.width, (self.contentScrollView.frame.
        size.height - keyboardHeight) + self.navigationController.toolbar.frame.size.height)];
}

/*  When the keyboard will hide from screen again prepare the user interface    */
- (void) keyboardWillHide:(NSNotification *) notification
{
    // reset the content scrollviews size
    [self.contentScrollView setFrame: CGRectMake(0, 0, self.view.frame.size.width, self.view.frame.size.height)];
}

#pragma mark    -
#pragma mark    UIActionSheet delegate

/*  when the actionsheet was dismissed with the given button index respond to it in this method */
- (void) actionSheet:(UIActionSheet *)actionSheet willDismissWithButtonIndex:(NSInteger)buttonIndex
{
    // when the cancel button was not pressed check which picker to show
    if (buttonIndex != [actionSheet cancelButtonIndex]) {

        // create the image picker controller and set the delegate object
        UIImagePickerController *imagePickerController = [[UIImagePickerController alloc] init];
        [imagePickerController setDelegate: self];
        [imagePickerController setMediaTypes: [NSArray arrayWithObject: (NSString *) kUTTypeImage]];

        // when the first index was pressed a new image should be taken
        if (buttonIndex == 0) {
```

```
            // configure for image capturing
            [imagePickerController setSourceType: UIImagePickerControllerSourceTypeCamera];
        } else if (buttonIndex == 1){

            // configure for selecting images
            [imagePickerController setSourceType: UIImagePickerControllerSourceTypeSavedPhotosAlbum];
        }

        // show the image picker controller
        [self presentViewController: imagePickerController animated: YES completion: ^{

            // notify navigation bar that the detail view should not be updated
            [(SBNavigationBar *)self.navigationController.navigationBar setShouldAcceptNextDetailView: NO];
        }];
    }
}

#pragma mark    -
#pragma mark    UITextView delegate

/*  When the text changes the textview should scroll to visible text    */
- (void) textViewDidChange:(UITextView *) textView
{
    // get the posotion of the last character in the textview
    UITextPosition *startPosition = [textView positionFromPosition: textView.endOfDocument offset: 0];

    // then get the textviews rectangle of the position of the last character
    CGRect textRect = [textView firstRectForRange: [textView textRangeFromPosition: startPosition toPosition: nil]];

    // get the scroll position for the content scroll container (includes the offset for container view and text
        view)
    CGFloat scrollToHeight = (textRect.size.height + 10);
    [self.contentScrollView scrollRectToVisible: CGRectMake(textRect.origin.x, textRect.origin.y, textRect.size.
        width, textRect.size.height + scrollToHeight) animated: YES];

    // scroll the textview to the selected range, the position of the cursor
    [textView scrollRangeToVisible: [textView selectedRange]];
}

/*  When the textview will start editing clear the placeholder text when necesary   */
- (BOOL) textViewShouldBeginEditing:(UITextView *) textView
{
    // when the placeholder is visible clear the textview
    if (self.isShowingPlaceholderText) {
        [textView setText: @""];
        self.isShowingPlaceholderText = NO;

        // set the typing font of the textview
        [textView setFont: self.textViewFont];
        [textView setTextColor: [UIColor blackColor]];
    }

    return YES;
}

/*  when the textview should end its editing and no text is inserted reset the placeholder  */
- (BOOL) textViewShouldEndEditing:(UITextView *) textView
{
    // when there is text inserted do nothing, otherwise reset the placeholder text
    if ([[textView text] length] == 0) {

        // set the placeholder font and color
        [textView setTextColor: [UIColor lightGrayColor]];
        [textView setFont: [UIFont italicSystemFontOfSize: 11]];

        // set the placeholder text and toolbar title when no image is set
        [textView setText: NSLocalizedString(@"optional.viewcontroller.textview.placeholder", nil)];

        if (self.imageDescriptor == nil) {
            [self.buttonView setTitle: NSLocalizedString(@"optional.viewcontroller.toolbar.skip.title", nil)
                forState: UIControlStateNormal];
        }
        self.isShowingPlaceholderText = YES;
    } else {

        // set the new title of the toolbar
        [self.buttonView setTitle: NSLocalizedString(@"optional.viewcontroller.toolbar.next.title", nil) forState:
            UIControlStateNormal];
    }

    return YES;
}

@end
```

```
//
//  SBPersonalDataViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 17-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBViewController.h"
#import "SBOperationDelegate.h"
#import "SBURLOperationDelegate.h"
#import "SBKeyboardToolbarDelegate.h"

@class SBTextField, SBKeyboardToolbar, SBUser;

@interface SBPersonalDataViewController : SBViewController <UITableViewDataSource, UITableViewDelegate,
    UIPickerViewDataSource, UIPickerViewDelegate, SBURLOperationDelegate, SBOperationDelegate, UITextFieldDelegate,
    SBKeyboardToolbarDelegate>

// the outlet tableview for personal details
@property (nonatomic, weak) IBOutlet UITableView *tableView;

// the current user object
@property (nonatomic, strong) SBUser *currentUser;

// sets whether the submit button should go to the location selection page or back to the main controller
@property (nonatomic) BOOL shouldPopOnSubmit;

@end
```

```
//
//  SBPersonalDataViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 17-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBPersonalDataViewController.h"
#import <QuartzCore/QuartzCore.h>
#import "SBNavigationBarDetailView.h"
#import "SBNavigationBar.h"
#import "SBTextField.h"
#import "SBURLOperation.h"
#import "SBOperationManager.h"
#import "SBAPIRequest.h"
#import "SBAPIResponse.h"
#import "SBCustomerDescriptor.h"
#import "SBKeyboardToolbar.h"
#import "SBKeyboardToolbarLayout.h"
#import "SBUser.h"
#import "SBReservationManager.h"
#import "SBViewCompatabilityFactory.h"

@interface SBPersonalDataViewController ()

// declare uneditable textfields
@property (nonatomic, strong) UITextField *townTextfield;
@property (nonatomic, strong) UITextField *streetTextField;
@property (nonatomic, strong)UITextField *corporationTextField;

// whether the corporation selector view is visible currently
@property (nonatomic) BOOL isCorporationSelectorVisible;
@property (nonatomic) BOOL currentAddressCredentialsIsValid;

// the array that holds all of the corporation choices
@property (nonatomic, strong) NSArray *corporations;
@property (nonatomic, strong) UIPickerView *corporationPicker;

// the current textfield that is selected and the keyboard toolbar
@property (nonatomic, weak) SBTextField *currentTextField;
@property (nonatomic, strong) SBKeyboardToolbar *keyboardToolbar;

// declare all private textfields that can be edited by the user
@property (nonatomic, strong) SBTextField *usernameTextfield;
@property (nonatomic, strong) SBTextField *emailTextField;
@property (nonatomic, strong) SBTextField *telephoneNumberTextField;
@property (nonatomic, strong) SBTextField *zipcodeTextField;
@property (nonatomic, strong) SBTextField *houseNumberAdditionTextField;

// this property identifies the UUID of the currently executing address lookup operation
@property (nonatomic, strong) NSUUID *currentOperationUUID;

// returns the corporation picker view and sets whether to show the loading indicator
- (BOOL) isCorporationPickerViewDataAvailable;
- (UIPickerView *) corporationPickerView;
- (void) setUpPickerView;

// returns a activity indicator view that will be shown in the center of the provided rectangle
- (UIActivityIndicatorView *) activatedActivityIndicatorViewForRect:(CGRect) rect;

// process the corporation array downloaded
- (void) processCorporations:(NSArray *) corporationArray;

// retrieve the button view that is shown in the corporation cell
- (UIView *) corporationCellSelectionIndicator;

// keyboard notification methods for responding to keyboard events
- (void) keyboardDidShow:(NSNotification *) notification;
- (void) keyboardWillHide:(NSNotification *) notification;

// get notified about text changes
- (void) textFieldTextDidChange:(NSNotification *) notification;

// when the user tappend the tableview background
- (void) shouldHideKeyboard:(UITapGestureRecognizer *) tapGesture;

// actions for when the user pressed the cancel or done button
- (IBAction) cancelButtonPressed:(UIBarButtonItem *) senderButton;
- (void) confirmButtonPressed:(UIBarButtonItem *) senderButton;

@end

// the different name identifiers for URL operations
static NSString *SBPersonalDataOperationCorporation     =   @"SBCorporationRequest";
static NSString *SBPersonalDataOperationAddress         =   @"SBAddressRequest";
```

```
// the corporation dictionary keys
static NSString *SBCorporationDictionaryIDKey           =   @"id";
static NSString *SBCorporationDictionaryDescriptionKey  =   @"description";

// dequeuable identifier
static NSString *SBPersonalReusableCellIdentifier       =   @"SBPersonalCellIdentifier";

// identifiers for the address lookup operatiopn
static NSString *SBPersonalAddressDictionaryStreetKey   =   @"street";
static NSString *SBPersonalAddressDictionaryTownKey     =   @"city";

@implementation SBPersonalDataViewController

/*  Initializatio method for when the controller is created via the storyboard  */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    // call super implementation
    self = [super initWithCoder: aDecoder];

    if (self) {

        // Custom initialization
        self.isCorporationSelectorVisible = NO;
        self.shouldPopOnSubmit = YES;
    }

    return self;
}

/*  When the view information was loaded in memory  */
- (void) viewDidLoad
{
    [super viewDidLoad];

    // set the controller title and the toolbar for this controller is visible
    [self setTitle: NSLocalizedString(@"data.viewcontroller.title", nil)];
    [self.navigationController setToolbarHidden: NO animated: YES];

    // create the request that loads the corporation from the API
    SBAPIRequest *corporationRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPIAllCustomersRequestType
        timeout: 120];
    NSDictionary *operationOptions = [NSDictionary dictionaryWithObject: SBPersonalDataOperationCorporation forKey:
        SBOperationNameOptionKey];
    SBURLOperation *corporationOperation = [[SBURLOperation alloc] initWithAPIRequest: corporationRequest options:
        operationOptions];

    // set the delegate and start the operation
    [corporationOperation setDelegate: self];
    [[SBOperationManager defaultManager] addOperation: corporationOperation];

    // add tap gesture recognizer to the tableview
    UITapGestureRecognizer *tapGestureRecognizer = [[UITapGestureRecognizer alloc] initWithTarget: self action:
        @selector(shouldHideKeyboard:)];
    [tapGestureRecognizer setCancelsTouchesInView: NO];
    [self.view addGestureRecognizer: tapGestureRecognizer];

    // create the save button
    UIButton *saveButton = [[UIButton alloc] initWithFrame: CGRectMake(160, 0, 160, 44)];
    [saveButton addTarget: self action: @selector(confirmButtonPressed:) forControlEvents:
        UIControlEventTouchUpInside];
    [saveButton setBackgroundColor: [UIColor colorWithRed: 0.25 green: 0.3203125 blue: 0.41015625 alpha: 1]];
    [[saveButton titleLabel] setFont: [UIFont systemFontOfSize: 13.0f]];
    [saveButton setTitle: NSLocalizedString(@"data.viewcontroller.toolbar.save.title", nil) forState:
        UIControlStateNormal];

    // create horizontal spacing between the toolbar buttons, the fixed space is used to get the toolbar right item
        against the right border
    UIBarButtonItem *fixedSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
        UIBarButtonSystemItemFixedSpace target: nil action: nil];
    fixedSpace.width -= 16.0f;

    // create the bar button from this save button and add that button to the toolbar
    UIBarButtonItem *barButtonSaveItem = [[UIBarButtonItem alloc] initWithCustomView: saveButton];
    [self setToolbarItems: [self.toolbarItems arrayByAddingObjectsFromArray: [NSArray arrayWithObjects:
        barButtonSaveItem, fixedSpace, nil]]];

    // initialize required textfields
    self.usernameTextfield = [[SBTextField alloc] initWithFrame: CGRectMake(10, 0, 260, 44)];
    self.emailTextField = [[SBTextField alloc] initWithFrame: CGRectMake(10, 0, 260, 44)];
    self.telephoneNumberTextField = [[SBTextField alloc] initWithFrame: CGRectMake(10, 0, 260, 44)];
    self.zipcodeTextField = [[SBTextField alloc] initWithFrame: CGRectMake(0, 0, 135, 44)];
    self.houseNumberAdditionTextField = [[SBTextField alloc] initWithFrame: CGRectMake(155, 0, 135, 44)];

    // initialize the non required textfields
    self.townTextfield = [[UITextField alloc] initWithFrame: CGRectMake(10, 0, 260, 44)];
    self.streetTextField = [[UITextField alloc] initWithFrame: CGRectMake(10, 0, 260, 44)];
    self.corporationTextField = [[UITextField alloc] initWithFrame: CGRectMake(10, 0, 260, 44)];
```

```
    [self.corporationTextField setEnabled: NO];
    [self.townTextfield setEnabled: NO];
    [self.streetTextField setEnabled: NO];

    // set all placeholder strings for the required textfields
    [self.usernameTextfield setPlaceholder: NSLocalizedString(@"data.viewcontroller.textfield.username.placeholder",
        nil)];
    [self.emailTextField setPlaceholder: NSLocalizedString(@"data.viewcontroller.textfield.email.placeholder", nil)]
        ;
    [self.telephoneNumberTextField setPlaceholder:
        NSLocalizedString(@"data.viewcontroller.textfield.telephone.placeholder", nil)];
    [self.zipcodeTextField setPlaceholder: NSLocalizedString(@"data.viewcontroller.textfield.zipcode.placeholder",
        nil)];
    [self.houseNumberAdditionTextField setPlaceholder:
        NSLocalizedString(@"data.viewcontroller.textfield.housenumber.placeholder", nil)];
    [self.streetTextField setPlaceholder: NSLocalizedString(@"data.viewcontroller.textfield.street.placeholder", nil
        )];
    [self.townTextfield setPlaceholder: NSLocalizedString(@"data.viewcontroller.textfield.town.placeholder", nil)];
    [self.corporationTextField setPlaceholder:
        NSLocalizedString(@"data.viewcontroller.textfield.corporation.placeholder", nil)];

    // configure each textfield to display the placeholder and input centered vertically
    [self.usernameTextfield setContentVerticalAlignment: UIControlContentVerticalAlignmentCenter];
    [self.emailTextField setContentVerticalAlignment: UIControlContentVerticalAlignmentCenter];
    [self.telephoneNumberTextField setContentVerticalAlignment: UIControlContentVerticalAlignmentCenter];
    [self.zipcodeTextField setContentVerticalAlignment: UIControlContentVerticalAlignmentCenter];
    [self.houseNumberAdditionTextField setContentVerticalAlignment: UIControlContentVerticalAlignmentCenter];
    [self.streetTextField setContentVerticalAlignment: UIControlContentVerticalAlignmentCenter];
    [self.townTextfield setContentVerticalAlignment: UIControlContentVerticalAlignmentCenter];
    [self.corporationTextField setContentVerticalAlignment: UIControlContentVerticalAlignmentCenter];

    // set each textfield's delegate
    [self.usernameTextfield setDelegate: self];
    [self.emailTextField setDelegate: self];
    [self.telephoneNumberTextField setDelegate: self];
    [self.zipcodeTextField setDelegate: self];
    [self.houseNumberAdditionTextField setDelegate: self];

    // set each textfield's next and previous textfield
    [self.usernameTextfield setNextTextField: self.emailTextField];
    [self.emailTextField setPreviousTextField: self.usernameTextfield];
    [self.emailTextField setNextTextField: self.telephoneNumberTextField];
    [self.telephoneNumberTextField setPreviousTextField: self.emailTextField];
    [self.telephoneNumberTextField setNextTextField: self.zipcodeTextField];
    [self.zipcodeTextField setPreviousTextField: self.telephoneNumberTextField];
    [self.zipcodeTextField setNextTextField: self.houseNumberAdditionTextField];
    [self.houseNumberAdditionTextField setPreviousTextField: self.zipcodeTextField];

    // set each textfield's corresponding indexpath
    [self.usernameTextfield setCorrespondingIndexPath: [NSIndexPath indexPathForRow: 0 inSection: 1]];
    [self.emailTextField setCorrespondingIndexPath: [NSIndexPath indexPathForRow: 0 inSection: 2]];
    [self.telephoneNumberTextField setCorrespondingIndexPath: [NSIndexPath indexPathForRow: 0 inSection: 3]];
    [self.zipcodeTextField setCorrespondingIndexPath: [NSIndexPath indexPathForRow: 0 inSection:4]];
    [self.houseNumberAdditionTextField setCorrespondingIndexPath: [NSIndexPath indexPathForRow: 0 inSection:4]];

    // add text description for each toolbar
    [self.usernameTextfield setToolbarInfo: NSLocalizedString(@"data.viewcontroller.keyboard.username.title", nil)];
    [self.emailTextField setToolbarInfo: NSLocalizedString(@"data.viewcontroller.keyboard.email.title", nil)];
    [self.telephoneNumberTextField setToolbarInfo: NSLocalizedString(@"data.viewcontroller.keyboard.telephone.title"
        , nil)];
    [self.zipcodeTextField setToolbarInfo: NSLocalizedString(@"data.viewcontroller.keyboard.zipcode.title", nil)];
    [self.houseNumberAdditionTextField setToolbarInfo:
        NSLocalizedString(@"data.viewcontroller.keyboard.housenumber.title", nil)];

    // create the custom accessory view that will be displayed on top of the keyboard
    self.keyboardToolbar = [[SBKeyboardToolbar alloc] init];
    [self.keyboardToolbar setDelegateObject: self];

    // set the input view of the editable textfields's
    [self.usernameTextfield setInputAccessoryView: self.keyboardToolbar];
    [self.emailTextField setInputAccessoryView: self.keyboardToolbar];
    [self.telephoneNumberTextField setInputAccessoryView: self.keyboardToolbar];
    [self.zipcodeTextField setInputAccessoryView: self.keyboardToolbar];
    [self.houseNumberAdditionTextField setInputAccessoryView: self.keyboardToolbar];

    // append the capitalization type of the keyboards
    [self.emailTextField setAutocapitalizationType: UITextAutocapitalizationTypeNone];
    [self.telephoneNumberTextField setAutocapitalizationType: UITextAutocapitalizationTypeNone];

    // set the input style
    [self.emailTextField setKeyboardType: UIKeyboardTypeEmailAddress];
    [self.telephoneNumberTextField setKeyboardType: UIKeyboardTypeNumbersAndPunctuation];

    // custom cinfiguration for the zipcode and housenumber field, baceasue the parent cell wil be clear colored
        this background should be white and the textfield layer should have rounded borders
    [self.zipcodeTextField setBackgroundColor: [UIColor whiteColor]];
    [self.houseNumberAdditionTextField setBackgroundColor: [UIColor whiteColor]];
```

```
    [[self.zipcodeTextField layer] setMasksToBounds: YES];
    [[self.houseNumberAdditionTextField layer] setMasksToBounds: YES];
    [[self.zipcodeTextField layer] setCornerRadius: 3.0f];
    [[self.houseNumberAdditionTextField layer] setCornerRadius: 3.0f];

    // also the text indentation should be set (custom method)
    [self.zipcodeTextField setTextIndentation: 10.0f];
    [self.houseNumberAdditionTextField setTextIndentation: 10.0f];

    // when the user was not set create a new one
    if (self.currentUser == nil) {
        self.currentUser = [SBUser sharedUser];

        // when fetched from the user defaults prefill the information
        if ([self.currentUser fetchFromUserDefaults]) {

            // prefill all textfields with existing information
            [self.usernameTextfield setText: [self.currentUser userFirstName]];
            [self.emailTextField setText: [self.currentUser userEmail]];
            [self.telephoneNumberTextField setText: [self.currentUser userTelephoneNumber]];
            [self.zipcodeTextField setText: [self.currentUser zipcode]];
            [self.houseNumberAdditionTextField setText: [self.currentUser houseNumber]];
            [self.streetTextField setText: [self.currentUser street]];
            [self.townTextfield setText: [self.currentUser town]];

            // when an addition is available append the text of the house number
            if ([[self.currentUser addition] length] > 0) {
                [self.houseNumberAdditionTextField setText: [NSString stringWithFormat: @"%@ %@", [self.currentUser
                    houseNumber], [self.currentUser addition]]];
            }

            // set text in  the corporation selector
            [self.corporationTextField setText: [[self.currentUser userCustomer] customerName]];

            // when the street field does not hold nil set valid address
            if (self.streetTextField != nil) {
                self.currentAddressCredentialsIsValid = YES;
            }
        }
    }

    // register for the keyboard hiding and showing notifications
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboardDidShow:) name:
        UIKeyboardDidShowNotification object: nil];
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboardWillHide:) name:
        UIKeyboardWillHideNotification object: nil];

    // get notified when the text changes for the zipcode and housenumber textfields
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(textFieldTextDidChange:) name:
        UITextFieldTextDidChangeNotification object: self.zipcodeTextField];
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(textFieldTextDidChange:) name:
        UITextFieldTextDidChangeNotification object: self.houseNumberAdditionTextField];
}

/*  Prepare the interface when the view will come on screen */
- (void) viewWillAppear:(BOOL)animated
{
    // call super implementatiom
    [super viewWillAppear: animated];

    // when the tabelview does not have a superview the view must be added to screen
    if ([self.tableView superview] == nil) {
        [self.tableView setFrame: CGRectMake(10, 0, 300, self.view.frame.size.height)];
        [self.view addSubview: self.tableView];
    }

    // create the new navigation bar detail view
    SBNavigationBarDetailView *detailView = [[SBNavigationBarDetailView alloc] initWithDetailText: NSLocalizedString
        (@"data.viewcontroller.navigation.description", nil)];

    // set the new detail view
    SBNavigationBar *navigationBar = (SBNavigationBar *)[self.navigationController navigationBar];
    [navigationBar setDetailView: detailView];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

/*  Returns an activated activity indicator view for the center of the provided rect    */
- (UIActivityIndicatorView *) activatedActivityIndicatorViewForRect:(CGRect) rect
{
    // create the activity indicator
    UIActivityIndicatorView *activityIndicatorView = [[UIActivityIndicatorView alloc]
        initWithActivityIndicatorStyle: UIActivityIndicatorViewStyleGray];
```

```
    // configure the activity indicator view by setting frame and center
    [activityIndicatorView setFrame: rect];
    [activityIndicatorView setCenter: CGPointMake((CGRectGetWidth(rect) / 2), (CGRectGetHeight(rect) / 2))];
    [activityIndicatorView startAnimating];

    // return the activity indicator
    return activityIndicatorView;
}

/*  Retrieve the view which indicates that the corporation view is selectable   */
- (UIView *) corporationCellSelectionIndicator
{
    // create the image view of the provided image
    UIImageView *corporationImageView = [[UIImageView alloc] initWithImage: [UIImage imageNamed:
        @"WoningCorporatieCellSelectie"]];

    // get the image size height and width
    CGFloat imageHeight = corporationImageView.image.size.height;
    CGFloat imageWidth = corporationImageView.image.size.width;

    // configure the frame and dimensions of the image view
    [corporationImageView setFrame: CGRectMake((300 - (imageWidth + 10)), (44 - imageHeight) / 2, imageWidth,
        imageHeight)];
    return corporationImageView;
}

/*  When the user tappednthe background view of the tableview hide the keyboard when visible    */
- (void) shouldHideKeyboard:(UITapGestureRecognizer *) tapGesture
{
    // when a current textfield is enabled make it resign first responder
    if (self.currentTextField != nil && [self.currentTextField isFirstResponder]) {
        [self.currentTextField resignFirstResponder];
    }
}

/*  When the user presses the cancel button the pop action should be activated  */
- (IBAction) cancelButtonPressed:(UIBarButtonItem *) senderButton
{
    // when a textfield is shown resign it first responder
    if (self.currentTextField != nil && [self.currentTextField isFirstResponder]) {
        [self.currentTextField resignFirstResponder];
    }

    // activate pop action on the navigation controller
    [self.navigationController popViewControllerAnimated: YES];
}

/*  When the confirm button is pressed validate all input and save when possible    */
- (void) confirmButtonPressed:(UIBarButtonItem *) senderButton
{
    // set whether all data is valid
    BOOL isDataValid = YES;

    // when the address validation or other fields are not valid
    if (!self.currentAddressCredentialsIsValid) {

        // mark the address information as invalid
        [self.zipcodeTextField setTextColor: [UIColor redColor]];
        [self.houseNumberAdditionTextField setTextColor: [UIColor redColor]];
        isDataValid = NO;
    }

    // when the firstname field is invalid mark it that way
    if (![SBUser validateFullNameString: [self.usernameTextfield text]]) {
        [self.usernameTextfield setIsMarkedAsInvalid: YES];
        isDataValid = NO;
    }

    // validate the email text
    if (![SBUser isValidEmail: [self.emailTextField text]]) {
        [self.emailTextField setIsMarkedAsInvalid: YES];
        isDataValid = NO;
    }

    // when the telephone number is invalid mark it that way
    if (![SBUser validatePhoneNumber: [self.telephoneNumberTextField text]]) {
        [self.telephoneNumberTextField setIsMarkedAsInvalid: YES];
        isDataValid = NO;
    }

    // when no customer is selected mark that textfield as invalid
    if ([[self corporationPickerView] selectedRowInComponent: 0] == -1) {
        [self.corporationTextField setTextColor: [UIColor redColor]];
        isDataValid = NO;
    }
```

```
    // when everything is validated save to the user
    if (isDataValid) {

        // extract all parts of the address using container
        NSDictionary *addressContainer = nil;

        // generate the address parts
        if (![SBUser isValidZipCode: [self.zipcodeTextField text] withHouseNumber: [self.
            houseNumberAdditionTextField text] results: &addressContainer]) {
            return;
        }

        // set the current user information that is provided in the textfields
        [self.currentUser setUserCustomer: [self.corporations objectAtIndex: [[self corporationPickerView]
            selectedRowInComponent: 0]]];
        [self.currentUser setUserEmail: [self.emailTextField text]];
        [self.currentUser setUserFirstName: [self.usernameTextfield text]];
        [self.currentUser setUserTelephoneNumber: [self.telephoneNumberTextField text]];
        [self.currentUser setStreet: [self.streetTextField text]];
        [self.currentUser setTown: [self.townTextfield text]];
        [self.currentUser setZipcode: [addressContainer objectForKey: SBUserValidationResultKeyZipCode]];
        [self.currentUser setHouseNumber: [addressContainer objectForKey: SBUserValidationResultKeyHouseNumber]];
        [self.currentUser setAddition: [addressContainer objectForKey: SBUserValidationResultKeyAddition]];

        // save the new data
        [self.currentUser save];

        // pop to the previous view controller
        if (self.shouldPopOnSubmit) {

            // no next view controller should be shown, cancel the new reservation
            [[SBReservationManager defaultReservationManager] didCancelReservation];
            [self.navigationController popViewControllerAnimated: YES];
        } else {

            // the location view controller should be shown
            [self performSegueWithIdentifier: @"userDidEnterValidCredentials:" sender: self];
        }
    }
}

#pragma mark    -
#pragma mark    UITableView datasource & delegate

/*  Returns the amount of section in the tableview  */
- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView
{
    return 7;
}

/*  Returns the amount of rows per section  */
- (NSInteger) tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    // when the first section is asked for and the corporation selector is visible return 2 cells
    if (section == 0 && self.isCorporationSelectorVisible) {
        return 2;
    } else {
        return 1;
    }
}

/*  Returns the cell for a specific indexpath in the tableview  */
- (UITableViewCell *) tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // create a new cell with no reuse identifier
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier: SBPersonalReusableCellIdentifier];

    if (cell == nil) {

        // create a new usable cell with reuse identifier
        cell = [[UITableViewCell alloc] initWithStyle: UITableViewCellStyleDefault reuseIdentifier:
            SBPersonalReusableCellIdentifier];
    } else {

        // loop through the subviews of the content view and remove textfields etc
        for (UIView *subview in [cell.contentView subviews]) {

            // only remove textfields and picker views
            if (![subview isKindOfClass: NSClassFromString(@"UITableViewCellContentView")]) {
                [subview removeFromSuperview];
            }
        }

        // reset the background color
        [cell setBackgroundColor: [UIColor whiteColor]];
    }
```

```
    // check for which section to display the cell of
    switch ([indexPath section]) {
        case 0:

            if ([indexPath row] == 0) {

                // for selecting a new hirer, add the informative texfield for corporations
                [cell.contentView addSubview: self.corporationTextField];
                [cell.contentView addSubview: [self corporationCellSelectionIndicator]];
            } else {

                // when the picker view information is available show the pickerview, otherwise show loading
                    indicator
                if ([self isCorporationPickerViewDataAvailable]) {

                    // add the picker view to the cell
                    [cell.contentView addSubview: [self corporationPickerView]];
                    [self setUpPickerView];
                } else {

                    // add the activity indicator to the view
                    [cell.contentView addSubview: [self activatedActivityIndicatorViewForRect: CGRectMake(0, 0, 300,
                        44)]];
                }
            }
            break;

        case 1:
            // the username cell, add the username textfield to the cell
            [cell.contentView addSubview: self.usernameTextfield];
            break;

        case 2:
            // the email cell, add the email textfield to the cell
            [cell.contentView addSubview: self.emailTextField];
            break;

        case 3:
            // the telephone number cell, add the telephone number textfield
            [cell.contentView addSubview: self.telephoneNumberTextField];
            break;

        case 4:
            // the zipcode and house number textfield cell, this cell should be transparant
            [cell setBackgroundColor: [UIColor clearColor]];
            [cell.contentView addSubview: self.zipcodeTextField];
            [cell.contentView addSubview: self.houseNumberAdditionTextField];
            break;

        case 5:
            // the street or address textfield cell, disabled by default
            [cell.contentView addSubview: self.streetTextField];
            break;

        case 6:
            // the town name of the found address
            [cell.contentView addSubview: self.townTextfield];
            break;

        default:
            break;
    }

    // no cell can be selected
    [cell setSelectionStyle: UITableViewCellSelectionStyleNone];

    return cell;
}

/*  When the cell will be displayed and the corresponding textfield is not yet the first responder this method will
    take care of it */
- (void) tableView:(UITableView *)tableView willDisplayCell:(UITableViewCell *)cell forRowAtIndexPath:(NSIndexPath *
    )indexPath
{
    // when the current textfield should be visible, it is not and this is the cell that includes that textfield
        make it become first responder
    if (self.currentTextField != nil && ![self.currentTextField isFirstResponder] && [[self.currentTextField
        correspondingIndexPath] isEqual: indexPath]) {
        [self.currentTextField becomeFirstResponder];
    }
}

/*  When a row is selected take spceific action via this method */
- (void) tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    // the corporation cell has been selected, toggle the corporation selector
    if ([indexPath section] == 0 && [indexPath row] == 0) {
```

```
        // toggle the visible boolean and reload the first section
        self.isCorporationSelectorVisible = !self.isCorporationSelectorVisible;
        [tableView reloadSections: [NSIndexSet indexSetWithIndex: 0] withRowAnimation:
            UITableViewRowAnimationAutomatic];

        // when a textfield is selected hide the keyboard when first responder
        if (self.currentTextField != nil && [self.currentTextField isFirstResponder]) {
            [self.currentTextField resignFirstResponder];
        }
    }
}

/*  Override the default tableview height   */
- (CGFloat) tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // the corporation selection cell is higher than the other textfield cells, because a picker view is built in
        this view
    if ([indexPath section] == 0 && [indexPath row] == 1) {

        // when the picker view is available to be shown return the height of this picker view
        if ([self isCorporationPickerViewDataAvailable]) {
            return 216;
        } else {

            // return a smaller height for only the activity indicator
            return 44;
        }

    } else {
        return 44;
    }
}

/*  Override the default footer height between sections */
- (CGFloat) tableView:(UITableView *)tableView heightForFooterInSection:(NSInteger)section
{
    // for the last cell add extra footer space
    if (section == 6) {
        return 10;
    } else {
        return 3;
    }
}

/*  Override the header height espacially for the headers without text strings  */
- (CGFloat) tableView:(UITableView *)tableView heightForHeaderInSection:(NSInteger)section
{
    // for headers with title a higher header should be drawn
    if (section == 0 || section == 1 || section == 4) {
        return 40;
    } else {
        return 3;
    }
}

/*  Returns the title for the tableview section provided    */
- (NSString *) tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section
{
    // switch to find the correct string for section
    if (section == 0) {
        return NSLocalizedString(@"data.viewcontroller.tableview.corporation.title", nil);
    } else if (section == 1) {
        return NSLocalizedString(@"data.viewcontroller.tableview.personal.title", nil);
    } else if (section == 4) {
        return NSLocalizedString(@"data.viewcontroller.tableview.address.title", nil);
    } else return nil;
}

#pragma mark    -
#pragma mark    Corporation Delegation, Datasource & other

/*  start to process all corporations downloaded to the actual array    */
- (void) processCorporations:(NSArray *) corporationArray
{
    // create the mutable array for corporations and number formatter for the ID objects
    NSMutableArray *tempCorporations = [NSMutableArray arrayWithCapacity: [corporationArray count]];
    NSNumberFormatter *numberFormatter = [[NSNumberFormatter alloc] init];

    // enumerate through all corporations in the array
    for (NSDictionary *corporationDictionary in corporationArray) {

        // create default customer ID
        NSNumber *corporationId = nil;

        // get the ID object formatted as a number from the dictionary
        if (![[corporationDictionary objectForKey: SBCorporationDictionaryIDKey] isKindOfClass: [NSNumber class]]) {
```

```
            corporationId = [numberFormatter numberFromString: [corporationDictionary objectForKey:
                SBCorporationDictionaryIDKey]];
        } else {

            // the ID is already formatted as NSNumber, just set the object
            corporationId = [corporationDictionary objectForKey: SBCorporationDictionaryIDKey];
        }

        // create a new customer descriptor from the ID and description object
        SBCustomerDescriptor *newCustomer = [[SBCustomerDescriptor alloc] initWithID: corporationId andName:
            [corporationDictionary objectForKey: SBCorporationDictionaryDescriptionKey]];

        // add new customer to the array
        [tempCorporations addObject: newCustomer];
    }

    // set the unmutable array corporations
    self.corporations = [NSArray arrayWithArray: tempCorporations];
}

/*  Returns whether the information for corporation picking is available for the user   */
- (BOOL) isCorporationPickerViewDataAvailable
{
    // depends on the corporation array
    return (self.corporations != nil);
}

/*  Returns the picker view that will be placed in the corresponding cell   */
- (UIPickerView *) corporationPickerView
{
    // when the picker is already created return it
    if (self.corporationPicker != nil) {
        return self.corporationPicker;
    }

    // retrieve a cell to find out its bounds
    CGRect actualBounds = [[[self.tableView cellForRowAtIndexPath: [NSIndexPath indexPathForRow: 0 inSection: 0]]
        contentView] bounds];

    // create the picker view with a zero height, this height can be read back later
    self.corporationPicker = [[UIPickerView alloc] initWithFrame: CGRectMake(0, 0, actualBounds.size.width, 0)];

    // configure the picker view
    [self.corporationPicker setShowsSelectionIndicator: YES];
    [self.corporationPicker setDataSource: self];
    [self.corporationPicker setDelegate: self];
    [self.corporationPicker setBackgroundColor: [UIColor clearColor]];

    // select the middle corporation
    [self.corporationPicker selectRow: (self.corporations.count / 2) inComponent: 0 animated: NO];

    // return the created corporation picker
    return self.corporationPicker;
}

/*  Datasource method for getting the amount of items to show in the picker view    */
- (NSInteger) pickerView:(UIPickerView *) pickerView numberOfRowsInComponent:(NSInteger) component
{
    // return the amount of corporations
    return [self.corporations count];
}

/*  Datasource method to return the amount of components in the picker view */
- (NSInteger) numberOfComponentsInPickerView:(UIPickerView *) pickerView
{
    // picker view has only one component
    return 1;
}

/*  Returns the title that will be used for the picker view item at the specified component row */
- (NSString *) pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row forComponent:(NSInteger)component
{
    return [[self.corporations objectAtIndex: row] customerName];
}

/*  Returns the row height for the picker view item at specified location   */
- (CGFloat) pickerView:(UIPickerView *)pickerView rowHeightForComponent:(NSInteger)component
{
    return 44;
}

/*  When a picker view item is selected by the user this method is called   */
- (void) pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row inComponent:(NSInteger)component
{
    // set the textfield corporation text to the one specified in the row index
    [self.corporationTextField setText: [[self.corporations objectAtIndex: row] customerName]];
}
```

```
- (void) setUpPickerView
{
    // only append the background of the picker when on iOS 6.0
    if ([SBViewCompatabilityFactory cocoaVersion] == UICocoaVersionIdentifier_iOS_6_0) {

        // create background view
        /*UIView *pickerBackground = [[UIView alloc] initWithFrame: [self.corporationPicker frame]];
        [pickerBackground setBackgroundColor: [UIColor whiteColor]];

        // add the background view to the picke
        NSLog(@"%@", [self.corporationPicker subviews]);

        [[[self.corporationPicker subviews] objectAtIndex: 0] setBackgroundColor: [UIColor clearColor]];
        [[[self.corporationPicker subviews] objectAtIndex: 2] addSubview: pickerBackground];

        // remove unnecesary subviews
        [[[self.corporationPicker subviews] objectAtIndex: 8] setHidden: YES];
        [[[self.corporationPicker subviews] objectAtIndex: 3] setHidden: YES];
        [[[self.corporationPicker subviews] objectAtIndex: 5] setHidden: YES];
        [[[self.corporationPicker subviews] objectAtIndex: 6] setHidden: YES];
        [[[self.corporationPicker subviews] objectAtIndex: 7] setHidden: YES];

        // append the frame of the picker tableview
        [[[self.corporationPicker subviews] objectAtIndex: 4] setBackgroundColor: [UIColor clearColor]];*/
    }
}

#pragma mark    -
#pragma mark    SBURLOperation Delegate

/*  When an operation failed to be executed with the provided error */
- (void) operation:(SBOperation *) operation didFailWithError:(NSError *) error
{
    // get the name of the operation
    NSString *operationName = [[operation options] objectForKey: SBOperationNameOptionKey];

    // check which operation failed
    if ([operationName isEqualToString: SBPersonalDataOperationCorporation]) {

        // the corporation request failed to be executed
    } else if ([operationName isEqualToString: SBPersonalDataOperationAddress]) {

        // an address lookup operation has failed to be executed
    }
}

/*  When the operation provided has finished executing with the provided API response   */
- (void) operation:(SBOperation *) operation didFinishWithResult:(SBAPIResponse *) response
{
    // get the name of the operation
    NSString *operationName = [[operation options] objectForKey: SBOperationNameOptionKey];

    // check which operation failed
    if ([operationName isEqualToString: SBPersonalDataOperationCorporation]) {

        // the corporation request has loaded its data successfully, check for OK response before starting to
            process the data
        if ([response responseStatusCode] == SBHTTPResponseCodeOK) {
            [self processCorporations: (NSArray *)[response resultDictionary]];
        }

        // create index path for the corporation cell
        NSIndexPath *corporationIndexPath = [NSIndexPath indexPathForRow: 1 inSection: 0];

        // reload the corporations cell when visible
        if ([[self.tableView indexPathsForVisibleRows] containsObject: corporationIndexPath]) {
            [self.tableView reloadRowsAtIndexPaths: [NSArray arrayWithObject: corporationIndexPath]
                withRowAnimation: UITableViewRowAnimationAutomatic];
        }

    } else if ([operationName isEqualToString: SBPersonalDataOperationAddress]) {

        // an address lookup operation has failed to be executed
        NSDictionary *addressDictionary = (NSDictionary *) [response resultDictionary];
        self.currentAddressCredentialsIsValid = YES;

        // set the new textfield values
        [self.streetTextField setText: [addressDictionary objectForKey: SBPersonalAddressDictionaryStreetKey]];
        [self.townTextfield setText: [addressDictionary objectForKey: SBPersonalAddressDictionaryTownKey]];
    }
}

#pragma mark    -
#pragma mark    UIKeyboard Notification & toolbar Delegate

/*  When the keyboard did become visible the view should be prepared for the new frame bounds   */
```

```
- (void) keyboardDidShow:(NSNotification *) notification
{
    // get the height of the keyboard that is shown
    CGFloat keyboardHeight = [[[notification userInfo] valueForKey: UIKeyboardFrameEndUserInfoKey] CGRectValue].size
        .height;

    // append the keyboard height to the tableview
    [self.tableView setFrame: CGRectMake(10, 0, 300, ((self.view.frame.size.height - keyboardHeight) + self.
        navigationController.toolbar.frame.size.height))];
}

/*  When the keyboard will start the hiding process already make sure the bounds of the view are recalculated   */
- (void) keyboardWillHide:(NSNotification *) notification
{
    // set the current textfiield to nil
    self.currentTextField = nil;

    // append the tableview height
    [self.tableView setFrame: CGRectMake(10, 0, 300, self.view.frame.size.height)];
}

/*  When the left button of the UIKeyboard toolbar is pressed this method is called */
- (void) keyboardToolbarLeftBarButtonTapped:(SBKeyboardToolbar *) keyboardToolbar
{
    // when a previous textfield is available make this textfield the first responder
    if ([self.currentTextField previousTextField] != nil) {

        // scroll to the corresponding textfield and make it first responder
        [self.tableView scrollToRowAtIndexPath: [[self.currentTextField previousTextField] correspondingIndexPath]
            atScrollPosition: UITableViewScrollPositionMiddle animated: YES];
        self.currentTextField = [self.currentTextField previousTextField];

        // when the cell is already visible become first responder
        if ([[self.tableView indexPathsForVisibleRows] containsObject: [self.currentTextField correspondingIndexPath
            ]]) {
            [self.currentTextField becomeFirstResponder];
        }
    }
}

/*  When the right keyboard toolbar button is pressed this method is called */
- (void) keyboardToolbarRightBarButtonTapped:(SBKeyboardToolbar *) keyboardToolbar
{
    // set next responder when available
    if ([self.currentTextField nextTextField] != nil) {

        // scroll to the corresponding textfield and make it first responder
        [self.tableView scrollToRowAtIndexPath: [[self.currentTextField nextTextField] correspondingIndexPath]
            atScrollPosition: UITableViewScrollPositionMiddle animated: YES];
        self.currentTextField  = [self.currentTextField nextTextField];

        // when the cell is already visible become first responder
        if ([[self.tableView indexPathsForVisibleRows] containsObject: [self.currentTextField correspondingIndexPath
            ]]) {
            [self.currentTextField becomeFirstResponder];
        }

    } else {

        // final cell is shown make it resign first responder
        [self.currentTextField resignFirstResponder];
    }
}

#pragma mark    -
#pragma mark    UITextField Delegate

/*  Respond to when the UITextField is being edited */
- (void) textFieldDidBeginEditing:(UITextField *) textField
{
    // set the current textfield
    self.currentTextField = (SBTextField *)textField;
    SBKeyboardToolbarLayout *newLayout = nil;

    // mark as valid
    [self.currentTextField setIsMarkedAsInvalid: NO];

    // check what style of toolbar to show on top of the keyboard
    if ([self.currentTextField nextTextField] != nil && [self.currentTextField previousTextField] != nil) {

        // show a toolbar with next & previous buttons
        newLayout = [SBKeyboardToolbarLayout layoutWithLeftPreviousButtonRightNextButton];
        [newLayout setTitle: [self.currentTextField toolbarInfo]];

    } else if ([self.currentTextField nextTextField] != nil && [self.currentTextField previousTextField] == nil) {

        // show a toolbar with only the next button enabled
```

```
        UIBarButtonItem *nextBarButtonItem = [[UIBarButtonItem alloc] initWithTitle: NSLocalizedString(@"Next", nil)
            style: UIBarButtonItemStylePlain target: nil action: nil];
        newLayout = [SBKeyboardToolbarLayout layoutWithLeftBarButtonItem: nil rightBarButtonItem: nextBarButtonItem
            title: [self.currentTextField toolbarInfo]];
    } else {

        // show the done toolbar state
        UIBarButtonItem *previousBarButtonItem = [[UIBarButtonItem alloc] initWithTitle:
            NSLocalizedString(@"Previous", nil) style: UIBarButtonItemStylePlain target: nil action: nil];
        UIBarButtonItem *doneBarButtonItem = [[UIBarButtonItem alloc] initWithTitle: NSLocalizedString(@"Done", nil)
            style: UIBarButtonItemStyleDone target: nil action: nil];
        newLayout = [SBKeyboardToolbarLayout layoutWithLeftBarButtonItem: previousBarButtonItem rightBarButtonItem:
            doneBarButtonItem title: [self.currentTextField toolbarInfo]];
    }

    // set the new toolbar layout info and let the toolbar redraw its content
    [self.keyboardToolbar redrawWithLayout: newLayout];

    // when the corporation selection is currently shown it should be hidden
    if (self.isCorporationSelectorVisible) {

        // set to false and reload the first section
        self.isCorporationSelectorVisible = NO;
        [self.tableView reloadSections: [NSIndexSet indexSetWithIndex: 0] withRowAnimation:
            UITableViewRowAnimationAutomatic];
    }
}

/*  when the text changes in the zipcode or house number field this method gets called  */
- (void) textFieldTextDidChange:(NSNotification *) notification
{
    // reset the address validation
    self.currentAddressCredentialsIsValid = NO;

    // when a current request is being executed cancel it
    if (self.currentOperationUUID != nil) {
        [[SBOperationManager defaultManager] cancelOperations: [NSArray arrayWithObject: self.currentOperationUUID]]
            ;
    }

    // when the zipcode is changed strip any spaces of the string
    if ([[notification object] isEqual: self.zipcodeTextField]) {
        [self.zipcodeTextField setText: [[self.zipcodeTextField text] stringByTrimmingCharactersInSet:
            [NSCharacterSet whitespaceAndNewlineCharacterSet]]];
    }

    // create the container dictionary for results
    NSDictionary *addressContainer = nil;

    // validate whether the given values are currently valid to be checked via the API database
    if ([SBUser isValidZipCode: [self.zipcodeTextField text] withHouseNumber: [self.houseNumberAdditionTextField
        text] results: &addressContainer]) {

        // set the new validated values in the textfield
        [self.zipcodeTextField setText: [addressContainer objectForKey: SBUserValidationResultKeyZipCode]];
        [self.houseNumberAdditionTextField setText: [addressContainer objectForKey:
            SBUserValidationResultKeyMergedNumber]];

        // create the API request for zipcode checking
        SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPIAddressLookupRequestType timeout:
            30];
        [APIRequest setPOSTValue: [addressContainer objectForKey: SBUserValidationResultKeyZipCode] forKey:
            @"zipcode"];
        [APIRequest setPOSTValue: [addressContainer objectForKey: SBUserValidationResultKeyHouseNumber] forKey:
            @"house_number"];
        [APIRequest setPOSTValue: [addressContainer objectForKey: SBUserValidationResultKeyAddition] forKey:
            @"addition"];

        // create options for the request
        NSDictionary *operationInfo = [NSDictionary dictionaryWithObject: SBPersonalDataOperationAddress forKey:
            SBOperationNameOptionKey];

        // create the operation that will be executed
        SBURLOperation *addressOperation = [[SBURLOperation alloc] initWithAPIRequest: APIRequest options:
            operationInfo];
        [addressOperation setDelegate: self];

        // start the operation
        self.currentOperationUUID = [[SBOperationManager defaultManager] addOperation: addressOperation];
    }
}

@end
```

```
//
//  SBPhoneCustomerViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBURLOperation.h"
#import "SBOperationManager.h"
#import "SBCustomerControllerDelegate.h"
#import "SBOperationDelegate.h"

@interface SBPhoneCustomerViewController : UIViewController <SBOperationDelegate, SBURLOperationDelegate,
    UITableViewDataSource, UITableViewDelegate, UISearchBarDelegate>

// the delegate object
@property (nonatomic, weak) id<SBCustomerControllerDelegate> delegate;
@property (nonatomic, strong) UITableView *customerTableView;
@property (nonatomic, strong) UISearchBar *searchBar;
@property (nonatomic) BOOL isSearchBarVisible;
@property (nonatomic) BOOL isLoadingCustomers;

// the array with all customers
@property (nonatomic, strong) NSMutableArray *customers;
@property (nonatomic, strong) NSMutableArray *searchedCustomers;

// declare methods activated by the user
- (IBAction) searchButtonTapped:(UIBarButtonItem *) sender;
- (IBAction) donebuttonTapped:(UIBarButtonItem *) sender;

@end
```

```
//
//  SBPhoneCustomerViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBPhoneCustomerViewController.h"
#import "SBCustomerDescriptor.h"
#import "SBErrorView.h"
#import "SBVerificationManager.h"
#import "SBAPIResponse.h"
#import "SBAPIRequest.h"


@interface SBPhoneCustomerViewController ()

// keyboard notification methods for showing and hiding
- (void) keyboardWillShow:(NSNotification *) notification;
- (void) keyboardWillHide:(NSNotification *) notification;

// removes all subviews that are allowed to be removed
- (void) removeAllSubviews;

@end

@implementation SBPhoneCustomerViewController

/*  Initializes the customer selection controller and registers for notifications   */
- (id) initWithCoder:(NSCoder *)aDecoder
{
    self = [super initWithCoder: aDecoder];

    if (self) {

        // create data arrays
        self.customers = [[NSMutableArray alloc] init];
        self.searchedCustomers = [[NSMutableArray alloc] init];

        // get notified about keyboard notifications
        [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboardWillShow:) name:
            UIKeyboardWillShowNotification object: nil];
        [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboardWillHide:) name:
            UIKeyboardWillHideNotification object: nil];
    }

    return self;
}

/*  When removed from memory remove this observer form the notification center  */
- (void) dealloc
{
    // stop retreiving notifications from the nc
    [[NSNotificationCenter defaultCenter] removeObserver: self];
}

/*  When the view was loaded initially from memory do initialization stuff  */
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
    [self setTitle: NSLocalizedString(@"customer.controller.title", nil)];

    // create the buttons for the navigation bar
    UIBarButtonItem *searchButton = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
        UIBarButtonSystemItemSearch target: self action: @selector(searchButtonTapped:)];
    UIBarButtonItem *donebutton = [[UIBarButtonItem alloc] initWithTitle:
        NSLocalizedString(@"customer.controller.navigation.done.title", nil) style: UIBarButtonItemStyleDone target:
        self action: @selector(donebuttonTapped:)];

    // add the buttons to the bar
    [self.navigationItem setLeftBarButtonItem: searchButton];
    [self.navigationItem setRightBarButtonItem: donebutton];

    // create the search bar
    self.searchBar = [[UISearchBar alloc] initWithFrame: CGRectZero];

    // configure the search bar
    [self.searchBar setAutocapitalizationType: UITextAutocapitalizationTypeNone];
    [self.searchBar setAutocorrectionType: UITextAutocorrectionTypeNo];
    [self.searchBar setDelegate: self];
}

/*  When a memory warning is received try to free up some memory    */
- (void) didReceiveMemoryWarning
{
```

```
    // this method has no memory warning implementation because this is the only view in a navigation controller, we
        do not want to clean up the memory because it is the first view the user will see when going back to this
        application.
    [super didReceiveMemoryWarning];
}

/*  Removes all subviews that are not needed    */
- (void) removeAllSubviews
{
    // loop through the subview array
    for (UIView *subview in [self.view subviews]) {

        // only remove tableviews and errorviews
        if ([subview isKindOfClass: [UITableView class]] || [subview isKindOfClass: [SBErrorView class]]) {
            [subview removeFromSuperview];
        }
    }
}

/*  When the view will appear check for needing to load the customers   */
- (void) viewWillAppear:(BOOL)animated
{
    // create the tableview when not already created
    if (self.customerTableView == nil) {

        // create the table view
        self.customerTableView = [[UITableView alloc] initWithFrame: CGRectZero style: UITableViewStylePlain];

        // configure the tableview
        [self.customerTableView setDelegate: self];
        [self.customerTableView setDataSource: self];
    }

    // only reload when not already loading
    if (!self.isLoadingCustomers) {

        // create request API object
        SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPIAllCustomersRequestType timeout:
            60];

        // create operation and set the delegate
        SBURLOperation *customerOperation = [[SBURLOperation alloc] initWithAPIRequest: APIRequest options: nil];
        [customerOperation setDelegate: self];

        // start the operation
        [[SBOperationManager defaultManager] addOperation: customerOperation];

        // create loading indicator view
        SBErrorView *loadingView = [[SBErrorView alloc] initWithFrame: self.view.bounds errorMessage:
            NSLocalizedString(@"customer.controller.loading.description", nil) withImageType:
            SBErrorImageTypeLoading];

        // add view to the screen
        [self.view addSubview: loadingView];
        [[self.navigationItem leftBarButtonItem] setEnabled: NO];
    }
}

/*  when the user wants to search through the list of customers this button gets tapped */
- (IBAction) searchButtonTapped:(UIBarButtonItem *)sender
{
    if (!self.isSearchBarVisible) {

        // set the frame of the search bar and the tableview
        [self.searchBar setFrame: CGRectMake(0, -44, 320, 44)];
        [self.view addSubview: self.searchBar];

        // animate the search bar in
        [UIView animateWithDuration: 0.25 animations: ^{

            // disable the left bar button
            [[self.navigationItem leftBarButtonItem] setEnabled: NO];

            // animate the search bar to the view and animate the tableview down
            [self.searchBar setFrame: CGRectMake(0, 0, 320, 44)];
            [self.customerTableView setFrame: CGRectMake(0, 44, self.view.bounds.size.width, self.view.bounds.size.
                height - 44)];

        } completion: ^(BOOL finished) {

            // enable the left bar button item and let the search bar become the first responder
            [[self.navigationItem leftBarButtonItem] setEnabled: YES];
            [self.searchBar becomeFirstResponder];
        }];

        // add the serach bar to the view
        self.isSearchBarVisible = YES;
```

```
        // reload the tableview to prevent a crash
        [self.customerTableView reloadData];
    }
    else {
        // resign first responder for the searchbar, this will hide the other views and recalculate
        [self.searchBar resignFirstResponder];
    }
}

/*  When the user is done with selecting the customer this method gets called   */
- (IBAction) donebuttonTapped:(UIBarButtonItem *)sender
{
    // message delegate that the picking finished
    if ([self.delegate respondsToSelector: @selector(customerControllerDidFinishPickingCustomer:)])
        [self.delegate customerControllerDidFinishPickingCustomer: self];
}

/*  when the keyboard will become visible this method gets called   */
- (void) keyboardWillShow:(NSNotification *)notification
{
    // get the new frame of the keyboard
    CGRect keyboardFrame = [[[notification userInfo] valueForKey: UIKeyboardFrameEndUserInfoKey] CGRectValue];

    // animate the tableview to change the frame accordingly for the keyboard
    [UIView animateWithDuration: [[[notification userInfo] valueForKey: UIKeyboardAnimationDurationUserInfoKey]
        floatValue] animations:^
    {
        // set the new tableview frame
        [self.customerTableView setFrame: CGRectMake(0, 44, self.view.bounds.size.width, self.view.bounds.size.
            height - (44 + keyboardFrame.size.height))];
    }];
}

/*  when the keyboard will hide this method gets called */
- (void) keyboardWillHide:(NSNotification *)notification
{
    // disable the search button
    [[self.navigationItem leftBarButtonItem] setEnabled: NO];

    // animate the searchbar away from the view and the tableview to full frame
    [UIView animateWithDuration: [[[notification userInfo] valueForKey: UIKeyboardAnimationDurationUserInfoKey]
        floatValue] animations:^
    {
        // set the frame of the views
        [self.customerTableView setFrame: self.view.bounds];
        [self.searchBar setFrame: CGRectMake(0, -44, 320, 44)];
    } completion: ^(BOOL finished)
    {
        // remove the searchbar from view
        [self.searchBar removeFromSuperview];
        self.isSearchBarVisible = NO;

        // reload the tableview
        [self.customerTableView reloadData];
        [[self.navigationItem leftBarButtonItem] setEnabled: YES];
    }];
}

#pragma mark    -
#pragma mark    SBOperation delegate methods

/*  Gets called when the operation fails to download the customer data  */
- (void) operation:(SBOperation *)operation didFailWithError:(NSError *)error
{
    // first remove all subviews to make room for the error view
    [self removeAllSubviews];

    // create the new error view and configure it
    SBErrorView *failView = [[SBErrorView alloc] initWithFrame: self.view.bounds errorMessage: [error
        localizedDescription] withImageType: SBErrorImageTypeFailed];

    // add the error view to the view
    [self.view addSubview: failView];
}

/*  When the customer data was successfully loaded this method gets called and will parse the data  */
- (void) operation:(SBOperation *) operation didFinishWithResult:(SBAPIResponse *) response
{
    // remove subviews
    [self removeAllSubviews];

    // when the success type is received continue
    if ([response objectResultType] == SBAPIResultSuccessType) {

        // the array of customers is held by the response
        NSArray *customers = (NSArray *)[response resultDictionary];
```

```
        [self.customers removeAllObjects];

        // create enumerator for fast enumeration thorugh the array
        NSEnumerator *customerEnumerator = [customers objectEnumerator];
        NSDictionary *currentCustomer = nil;

        // loop through all objects
        while (currentCustomer = [customerEnumerator nextObject]) {
            [self.customers addObject: [[SBCustomerDescriptor alloc] initWithID: [currentCustomer objectForKey:
                @"id"] andName: [currentCustomer objectForKey: @"description"]]];
        }

        // set the frame of the table view and add to the current view
        [self.customerTableView setFrame: self.view.bounds];
        [self.view addSubview: self.customerTableView];

        // enable the search button
        [[self.navigationItem leftBarButtonItem] setEnabled: YES];
    } else {

        // create the view configure it and add it to the view hierarchie
        SBErrorView *failView = [[SBErrorView alloc] initWithFrame: self.view.frame errorMessage:
            NSLocalizedString(@"customer.controller.load.error.description", nil) withImageType:
            SBErrorImageTypeWarning];
        [self.view addSubview: failView];
    }
}

#pragma mark    -
#pragma mark    UISearch bar delegate methods

/*  When the text in the search bar changed */
- (void) searchBar:(UISearchBar *)searchBar textDidChange:(NSString *)searchText
{
    // delete all objects
    [self.searchedCustomers removeAllObjects];

    // enumarate through all objects to find matching strings
    [self.customers enumerateObjectsUsingBlock: ^(id obj, NSUInteger idx, BOOL *stop) {

        // try to find the range of the string in the customers
        if ([[obj customerName] rangeOfString: searchText options: NSCaseInsensitiveSearch].location != NSNotFound){
            [self.searchedCustomers addObject: obj];
        }
    }];

    // reload the tableview
    [self.customerTableView reloadData];
}

#pragma mark    -
#pragma mark    UITableView delegate & datasource

/*  Resutns the amount of rows in the tableview */
- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

/*  Returns the amount of rows in a section */
- (NSInteger) tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    if (self.isSearchBarVisible){
        return [self.searchedCustomers count];
    }
    else {
        return [self.customers count];
    }
}

/*  Return a cell for each table view row   */
- (UITableViewCell *) tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // create the static string for reusing and try to retreive a reusable cell
    static NSString *cellIdentifier = @"CustomerIdentifierCell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier: cellIdentifier];

    // when the cell could not be reused create a new one
    if (cell == nil){
        cell = [[UITableViewCell alloc] initWithStyle: UITableViewCellStyleDefault reuseIdentifier: cellIdentifier];
    }

    // set the name of the customer in the cell
    if (self.isSearchBarVisible){
        [[cell textLabel] setText: [[self.searchedCustomers objectAtIndex: [indexPath row]] customerName]];
    }
    else {
```

```
        [[cell textLabel] setText: [[self.customers objectAtIndex: [indexPath row]] customerName]];
    }
    return cell;
}

/*  When a cell is selected message the delegate that a customer is selected    */
- (void) tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    SBCustomerDescriptor *selectedCustomer = nil;

    // get the object that is selected by the user, in search view or normal view
    if (self.isSearchBarVisible){
        selectedCustomer = [self.searchedCustomers objectAtIndex: [indexPath row]];
    }
    else {
        selectedCustomer = [self.customers objectAtIndex: [indexPath row]];
    }

    // check for the delegate to be able to respond
    if ([self.delegate respondsToSelector: @selector(customerController:didSelectItem:)]){
        [self.delegate customerController: self didSelectItem: selectedCustomer];
    }

}

@end
```

```
//
//  SBPhonePageDateViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 27-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <EventKit/EventKit.h>
#import "SBMaintenanceInterface.h"
#import "SBOperationManager.h"
#import "SBAvailableDateDescriptor.h"
#import "SBDayPartDescriptor.h"
#import "SBURLOperationDelegate.h"
#import "SBOperationDelegate.h"

@interface SBPhoneDateViewController : UIViewController <UITableViewDelegate, UITableViewDataSource,
    SBOperationDelegate, SBURLOperationDelegate, UIAlertViewDelegate>

// declare properties, the TableView holds the first available dates
@property (nonatomic, strong) UITableView *dateTableView;
@property (nonatomic, strong) NSMutableArray *possibleData;
@property (nonatomic, strong) NSIndexPath *selectedIndexPath;
@property (nonatomic, strong) NSDateFormatter *dateFormatter;
@property (nonatomic, strong) EKEventStore *eventStore;

@property (nonatomic) BOOL isLoadingAvailableData;

// method refreshes all dates possible
- (IBAction) nextButtonTapped:(UIBarButtonItem *) sender;
- (void) resetAllTimeIndication;

@end
```

```objc
//
//  SBPhonePageDateViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 27-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBPhoneDateViewController.h"
#import "SBErrorView.h"
#import "SBPhoneLoginViewController.h"
#import "SBPhoneOverviewViewController.h"
#import "SBVerificationManager.h"

#import "SBOperationManager.h"
#import "SBURLOperation.h"
#import "SBAPIRequest.h"
#import "SBAPIResponse.h"

@interface SBPhoneDateViewController()

// remove all views from the parent view
- (void) removeAllSubviews;

// refreshes all calendar events in the Calendar
- (void) refreshCalendars;
- (void) refreshLoadedDates;

// gets called when a calendar change notification is sent
- (void) calendarDidChangeNotification:(NSNotification *) notification;

// enters background and foreground
- (void) applicationDidEnterForeground:(NSNotification *) notification;

@end

@implementation SBPhoneDateViewController

/*  Initializes the controller with the given coder */
- (id) initWithCoder:(NSCoder *)aDecoder
{
    self = [super initWithCoder: aDecoder];

    if (self) {

        // create event store and date formatter
        self.eventStore = [[EKEventStore alloc] init];
        self.dateFormatter = [[NSDateFormatter alloc] init];

        // set locale and date format
        [self.dateFormatter setLocale: [NSLocale localeWithLocaleIdentifier: @"nl_NL"]];
        [self.dateFormatter setDateFormat: @"EEEE d MMMM"];

        // the date formatter TimeZone is fixed to Amsterdam TimeZone, is +2 hours (7200) after GMT
        [self.dateFormatter setTimeZone: [NSTimeZone timeZoneForSecondsFromGMT: 7200]];

        // create the date holding arrays
        self.selectedIndexPath = nil;
        self.possibleData = [[NSMutableArray alloc] initWithCapacity: 5];
        self.isLoadingAvailableData = NO;

        // observer the notification that the calendar data has changed
        [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(calendarDidChangeNotification:)
            name: EKEventStoreChangedNotification object: nil];

        // get notified when the application enters foreground and background
        [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(applicationDidEnterForeground:)
            name: UIApplicationDidBecomeActiveNotification object: nil];

    }
    return self;
}

/*  When removed from memory    */
- (void) dealloc
{
    // remove from notification center
    [[NSNotificationCenter defaultCenter] removeObserver: self];
}

/*  Setup the view when first loaded from memory    */
- (void)viewDidLoad
{
    [super viewDidLoad];

    // Do any additional setup after loading the view.
    [self setTitle: NSLocalizedString(@"date.controller.title", nil)];
```

```
    // create the done button
    UIBarButtonItem *rightButton = [[UIBarButtonItem alloc] initWithTitle:
        NSLocalizedString(@"date.controller.navigation.finish.title", nil) style: UIBarButtonItemStyleDone target:
        self action: @selector(nextButtonTapped:)];

    // initialy the done button is not enabled
    [self.navigationItem setRightBarButtonItem: rightButton];
    [[self.navigationItem rightBarButtonItem] setEnabled: NO];

    // get the status of this application wether it can access calendar events
    /*EKAuthorizationStatus calendarAuthorizationStatus = [EKEventStore authorizationStatusForEntityType:
        EKEntityTypeEvent];

    // when the authorization is not yet granted and asked show the dialog
    if (calendarAuthorizationStatus == EKAuthorizationStatusNotDetermined) {

        // request access to the EventKit, when allowed the calendar events are loaded for a specific date
        [self.eventStore requestAccessToEntityType: EKEntityTypeEvent completion: ^(BOOL granted, NSError *error) {

            // when access is granted data can be loaded for a selected date, when possible data is loaded the
                calendars should be refreshed
            if (granted && [self.possibleData count] > 0) {
                [self refreshCalendars];
            }
        }];
    }*/
}

/*  remove all subviews from its superview  */
- (void) removeAllSubviews
{
    // for each view remove it
    for (UIView *subview in [self.view subviews]){
        [subview removeFromSuperview];
    }
}

/*  When the view will appear on screen prepare the contents that will be shown to the user */
- (void) viewWillAppear:(BOOL)animated
{
    // when the tableview is not yet created
    if (self.dateTableView == nil) {

        // create the tableview
        self.dateTableView = [[UITableView alloc] initWithFrame: CGRectZero style: UITableViewStyleGrouped];

        // configure the tableview for delegate and datasource
        [self.dateTableView setDelegate: self];
        [self.dateTableView setDataSource: self];
    }

    // set tableview bounds
    [self.dateTableView setFrame: self.view.bounds];
    [self removeAllSubviews];

    // when the view appears, check for the application to not be busy loading the available dates and that it
        should refresh the available dates at the moment
    if (!self.isLoadingAvailableData){
        [self refreshLoadedDates];
    }
    else if (self.isLoadingAvailableData) {

        // add the indicator to the view and start animating
        UIView *loadingView = [[SBErrorView alloc] initWithFrame: self.view.bounds errorMessage:
            NSLocalizedString(@"date.controller.progress.loading.description", nil) withImageType:
            SBErrorImageTypeLoading];

        // add the loading indication to the main view
        [self.view addSubview: loadingView];
    }
}

/*  In this method tell the delegate to change its NavigationBar    */
- (void) viewDidAppear:(BOOL) animated
{
    // call super with method
    [super viewDidAppear: animated];
}

/*  Try to release some memory in case of a memory warning  */
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];

    // when the view is not currently shown in the window release objects
    if ([self.view window] == nil) {
```

```
        // remove all objects from the array because this data can be reloaded the next time the view is loaded,
            these dates do not remain valid for a long time in some cases
        [self.possibleData removeAllObjects];
        self.selectedIndexPath = nil;

        // remove all subviews from the main view and release the tableview
        [self removeAllSubviews];
        self.dateTableView = nil;
    }
}

/*  Gets this notification when the application will enter the background   */
- (void) applicationDidEnterForeground:(NSNotification *)notification
{
    // clear the view and remove selected objects
    self.selectedIndexPath = nil;
    [self.possibleData removeAllObjects];

    // refresh the loaded dates and delete all subviews
    [self removeAllSubviews];
    [self refreshLoadedDates];
}

/*  Refreshes all dates that are loaded from the server */
- (void) refreshLoadedDates
{
    // disable the next button
    [[self.navigationItem rightBarButtonItem] setEnabled: NO];
    self.selectedIndexPath = nil;

    // append booleans, so it appears that the application is loading the available dates
    self.isLoadingAvailableData = YES;

    // create the API request object
    SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPIAvailableDatesRequestType timeout: 60
        ];

    // create the operation and set its delegate object
    SBURLOperation *dateOperation = [[SBURLOperation alloc] initWithAPIRequest: APIRequest options: nil];
    [dateOperation setDelegate: self];

    // start the operation
    [[SBOperationManager defaultManager] addOperation: dateOperation];

    // add the indicator to the view and start animating
    UIView *loadingView = [[SBErrorView alloc] initWithFrame: self.view.bounds errorMessage:
        NSLocalizedString(@"date.controller.progress.loading.description", nil) withImageType:
        SBErrorImageTypeLoading];

    [self.view addSubview: loadingView];
}

/*  Resets all time indication objects  */
- (void) resetAllTimeIndication
{
    // reset all time indications
    for (int i = 0; i < [self.possibleData count]; i++)
         [[self.possibleData objectAtIndex: i] resetAllTimeIndication];
}

/*  When called the calendar events will be loaded from the event store, only when the authorization is granted */
- (void) refreshCalendars
{
    // get the authorization status of event kit
    EKAuthorizationStatus authorizationStatus = [EKEventStore authorizationStatusForEntityType: EKEntityTypeEvent];

    // when the application has access to calendar events continue
    if (authorizationStatus == EKAuthorizationStatusAuthorized && [self.possibleData count] > 0)
    {
        // create the predicate to find all events between certain dates and for the event entity type
        NSPredicate *eventPredicate = [self.eventStore predicateForEventsWithStartDate: [[self.possibleData
            objectAtIndex: 0] availableDate] endDate: [[self.possibleData objectAtIndex: ([self.possibleData count]
            - 1)] availableDate] calendars: [self.eventStore calendarsForEntityType: EKEntityTypeEvent]];

        // now fetch the events for the created predicate
        NSArray *calendarEvents = [self.eventStore eventsMatchingPredicate: eventPredicate];

        // when events are found process it with the day parts
        if ([calendarEvents count] > 0)
        {
            // create anumerators for the available dates & events, also create a calendar for the calculation of
                datecomponents
            NSEnumerator *dateEnumerator = [self.possibleData objectEnumerator];
            NSEnumerator *eventEnumerator = nil;
            SBAvailableDateDescriptor *currentAvailableDate = nil;
            EKEvent *currentEvent = nil;
```

```
            NSCalendar *gregorianCalendar = [[NSCalendar alloc] initWithCalendarIdentifier: NSGregorianCalendar];

            // loop through the available dates
            while (currentAvailableDate = [dateEnumerator nextObject])
            {
                // validate that each event exists in the date of the available date
                eventEnumerator = [calendarEvents objectEnumerator];

                // process each event in an available date descriptor
                while (currentEvent = [eventEnumerator nextObject])
                {
                    // make sure only events that are not cancelled are processed in the tableview
                    if ([currentEvent status] != EKEventStatusCanceled)
                        [currentAvailableDate processEvent: currentEvent withCalendar: gregorianCalendar];
                }
            }
        }
    }
}

/*  When a notification is received that indicates that the event store has changed respond to it   */
- (void) calendarDidChangeNotification:(NSNotification *)notification
{
    // when already fetched refresh all objects
    [self refreshCalendars];
    [self.dateTableView reloadData];
}

#pragma mark -
#pragma mark UITableView Delegate & Datasource

/*  table view has two sections, a selector part and a part that shows the users iCal agenda    */
- (NSInteger) numberOfSectionsInTableView:(UITableView *) tableView
{
    return 1;
}

/*  Returns the amount of rows for the current section  */
- (NSInteger) tableView:(UITableView *) tableView numberOfRowsInSection:(NSInteger) section
{
    // when a date is selected the tableview should expand to show the dayparts
    if (section == 0 && self.selectedIndexPath != nil)
        return [self.possibleData count] + [[self.possibleData objectAtIndex: [self.selectedIndexPath row]]
            totalAvailableTimes];

    // else when no date is selected just show the available dates
    if (section == 0)
        return [self.possibleData count];
    else return 0;
}

/*  Create a tableviewcell for the Indexpath asked for  */
- (UITableViewCell *) tableView:(UITableView *) tableView cellForRowAtIndexPath:(NSIndexPath *) indexPath
{
    // create a tableView cell with detail
    UITableViewCell *cell = [[UITableViewCell alloc] initWithStyle: UITableViewCellStyleSubtitle reuseIdentifier:
        nil];

    if (self.selectedIndexPath != nil) {

        // get the amount of time items in this index
        NSInteger currentItems = [[self.possibleData objectAtIndex: [self.selectedIndexPath row]]
            totalAvailableTimes] + 1;

        // this means that a row is selected to show times, check which rows
        // add all indexpaths that need to be reomved to the array
        for (int i = ([self.selectedIndexPath row] + 1), j = 0; i < ([self.selectedIndexPath row] + currentItems); i
            ++, j++)
        {
            // when this is the same, a time cell must be shown
            if ([indexPath isEqual: [NSIndexPath indexPathForRow: i inSection: 0]])
            {
                // get the date descriptor
                SBDayPartDescriptor *timeDescriptor = [[[self.possibleData objectAtIndex: [self.selectedIndexPath
                    row]] availableTime] objectAtIndex: j];
                UIFont *fontUsed = [UIFont systemFontOfSize: 10];

                // set selected when possible
                if ([timeDescriptor isSelected])
                {
                    fontUsed = [UIFont boldSystemFontOfSize: 14];
                    [cell setAccessoryType: UITableViewCellAccessoryCheckmark];
                }

                // set the font size for this cell
                [[cell textLabel] setFont: fontUsed];
                [[cell textLabel] setText: [timeDescriptor timeDescription]];
```

```
                // set the indentation level, this indicates a visible difference between dates and times
                [cell setIndentationWidth: 10];
                [cell setIndentationLevel: 1];
                [[cell detailTextLabel] setTextAlignment: NSTextAlignmentRight];

                return cell;
            }
        }

        // set the label text with the date
        NSInteger previousItems = [[self.possibleData objectAtIndex: [self.selectedIndexPath row]]
            totalAvailableTimes];
        NSInteger indexOfObject = 0;

        // when a row is selected with a higher row, the index has to be calculated, else the index 0 will be used
        if ([indexPath row] > previousItems)
            indexOfObject = ([indexPath row] - previousItems);

        [[cell textLabel] setText: [self.dateFormatter stringFromDate: [[self.possibleData objectAtIndex:
            indexOfObject] availableDate]]];
    }
    else
    {
        // set the dynamic preferred font and also text of the date identification
        [[cell textLabel] setFont: [UIFont boldSystemFontOfSize: 15.0f]];
        [[cell textLabel] setText: [self.dateFormatter stringFromDate: [[self.possibleData objectAtIndex: [indexPath
            row]] availableDate]]];
    }

    // no cell should show a selection style
    [cell setSelectionStyle: UITableViewCellSelectionStyleNone];
    return cell;
}

/*  When a row is selected by the user this method gets called  */
- (void) tableView:(UITableView *) tableView didSelectRowAtIndexPath:(NSIndexPath *) indexPath
{
    // delete the current 4 rows below the selected indexpath
    NSMutableArray *indexPathsToDelete = [[NSMutableArray alloc] init];
    NSMutableArray *indexPathsToInsert = [[NSMutableArray alloc] init];

    // check for the current indexpath to be selected
    if (self.selectedIndexPath != nil)
    {
        // get the amount of previous time items
        NSInteger previousItems = [[self.possibleData objectAtIndex: [self.selectedIndexPath row]]
            totalAvailableTimes] + 1;

        // add all indexpaths that need to be reomved to the array
        for (int i = ([self.selectedIndexPath row] + 1); i < ([self.selectedIndexPath row] + previousItems); i++)
            [indexPathsToDelete addObject: [NSIndexPath indexPathForRow: i inSection: 0]];

        // when an indexpath is selected and does not correspond to an indexpath of the dates, return
        for (int i = 0; i < [indexPathsToDelete count]; i++)
        {
            if ([indexPath isEqual: [indexPathsToDelete objectAtIndex: i]])
            {
                // get the selected date object
                SBDayPartDescriptor *descriptor = [[[self.possibleData objectAtIndex: [self.selectedIndexPath row]]
                    availableTime] objectAtIndex: i];

                // only reset the data when the current value of this object is NO, that way the isSelected value of
                    the other objects will be set to NO
                if (![descriptor isSelected])
                {
                    [self resetAllTimeIndication];
                    [descriptor setIsSelected: YES];

                    // enable the barbutton item for the next step, a date has been selected
                    [[self.navigationItem rightBarButtonItem] setEnabled: YES];
                }
                else
                {
                    // enable the barbutton item for the next step, a date has been selected
                    [[self.navigationItem rightBarButtonItem] setEnabled: NO];
                    [descriptor setIsSelected: NO];
                }
                // reload the selected cell
                [tableView reloadRowsAtIndexPaths: indexPathsToDelete withRowAnimation: UITableViewRowAnimationNone]
                    ;
                return;
            }
        }
    }

    // set the new _selected indexPath when a valid rows was selected
```

```
    if (![self.selectedIndexPath isEqual: indexPath])
    {
        // when rows are deleted, append the indexpath with the amount of rows that will be deleted. When the
            previous indexpath is lower than the current one append the new row with - the amount of deleted rows.
            This way the date cell will get to the correct IndexPath
        if ([indexPathsToDelete count] > 0 && [self.selectedIndexPath row] < [indexPath row]) {
            self.selectedIndexPath = [NSIndexPath indexPathForRow: ([indexPath row] - [indexPathsToDelete count])
                inSection: 0];
        }
        else self.selectedIndexPath = indexPath;

        // get the amount of time items to show
        NSInteger currentItems = [[self.possibleData objectAtIndex: [self.selectedIndexPath row]]
            totalAvailableTimes] + 1;

        // add all indexpaths that need to be inserted to the array
        for (int i = ([self.selectedIndexPath row] + 1); i < ([self.selectedIndexPath row] + currentItems); i++) {
            [indexPathsToInsert addObject: [NSIndexPath indexPathForRow: i inSection: 0]];
        }
    }
    else {
        self.selectedIndexPath = nil;
    }

    // following piece of code updates the tableView with the deleted rows (when available) and the inserted rows
        (when available). Wil be animated by the tableView
    // start updating the tableview
    [tableView beginUpdates];

    if ([indexPathsToDelete count] > 0)
        [tableView deleteRowsAtIndexPaths: indexPathsToDelete withRowAnimation: UITableViewRowAnimationTop];

    if ([indexPathsToInsert count] > 0)
        [tableView insertRowsAtIndexPaths: indexPathsToInsert withRowAnimation: UITableViewRowAnimationTop];

    // finish the updates
    [tableView endUpdates];
}

/*  Returns the Title for the header of a section   */
- (NSString *) tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section
{
    // the tableView contains one section
    return NSLocalizedString(@"date.controller.tableview.section.title", nil);
}

/*  Return the footerView for each section  */
- (CGFloat) tableView:(UITableView *)tableView heightForFooterInSection:(NSInteger)section
{
    return 0;
}

/*  When the next button is tapped this method gets called, will show the final controller  */
- (IBAction) nextButtonTapped:(UIBarButtonItem *)sender
{
    // show the alert view that indicates wether the defect should be posted
    UIAlertView *alertView = [[UIAlertView alloc] initWithTitle: NSLocalizedString(@"alertview.title.confirm", nil)
        message: NSLocalizedString(@"date.controller.alertview.confirm.description", nil) delegate: self
        cancelButtonTitle:NSLocalizedString(@"alertview.button.title.cancel", nil) otherButtonTitles:
        NSLocalizedString(@"alertview.button.title.confirm", nil), nil];

    // show the alert
    [alertView show];
}

#pragma mark    -
#pragma mark    UIAlertView delegate methods

/*  When a alertviw is dismissed decide what to do in this method   */
- (void) alertView:(UIAlertView *)alertView didDismissWithButtonIndex:(NSInteger)buttonIndex
{
    // when the user did not press the cancel button prepare for uploading
    if (buttonIndex != [alertView cancelButtonIndex]) {
        id<SBMaintenanceInterface> controller = (id<SBMaintenanceInterface>)[self.navigationController delegate];

        // commit the selected date and daypart objects, then message the delegate that the wizard is done with
            creating the reservation
        [controller viewController: self didChangeValue: [[self.possibleData objectAtIndex: [self.selectedIndexPath
            row]] availableDate] forProceedingKey: @"reservationDate"];
        [controller viewController: self didChangeValue: [[self.possibleData objectAtIndex: [self.selectedIndexPath
            row]] selectedDayPart] forProceedingKey: @"reservationDayPart"];

        // message done to the delegate
        [controller viewControllerDidFinishCreatingReservation: self];
    }
}
```

```
#pragma mark    --
#pragma mark    Connection delegate methods

/*  When the oepration failed to execute, show error in the view    */
- (void) operation:(SBOperation *)operation didFailWithError:(NSError *)error
{
    // remove subviews
    [self removeAllSubviews];

    // create the error view, automatically linges ut the error message and image to be centered
    SBErrorView *errorView = [[SBErrorView alloc] initWithFrame: self.view.bounds errorMessage: [error
        localizedDescription] withImageType: SBErrorImageTypeWarning];
    [self.view addSubview: errorView];

    // set that not loading data and reload on next view did appear
    self.isLoadingAvailableData = NO;
}

/*  When data is successfully loaded process it to check for errors */
- (void) operation:(SBOperation *) operation didFinishWithResult:(SBAPIResponse *) response
{
    // remove all subviews
    [self removeAllSubviews];

    // retrieve the date information to a dictionary
    NSArray *dateInformation = (NSArray *)[response resultDictionary];

    // when successfull parse the data
    if ([response objectResultType] == SBAPIResultSuccessType) {

        // delete all current date objects
        [self.possibleData removeAllObjects];

        // get the date value from the dictionary
        SBAvailableDateDescriptor *dateDescriptor = nil;

        // pasre the dates and add them to the possible dates array
        for (int i = 0; i < [dateInformation count]; i++) {
            dateDescriptor = [[SBAvailableDateDescriptor alloc] initWithDictionary: [dateInformation objectAtIndex:
                i]];
            [self.possibleData addObject: dateDescriptor];
        }

        // remove the activity indicator and show the tableview
        [self.view addSubview: self.dateTableView];
        [self.dateTableView reloadData];

    } else {

        // create the error view, automatically linges ut the error message and image to be centered
        SBErrorView *errorView = [[SBErrorView alloc] initWithFrame: self.view.bounds errorMessage:
            NSLocalizedString(@"date.controller.result.error.corrupted", nil) withImageType: SBErrorImageTypeWarning
            ];
        [self.view addSubview: errorView];
    }
}

#pragma mark    -

@end
```

```
//
//  SBPhonePageCategoryViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 26-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBCollectionController.h"
#import "Defect.h"

@interface SBPhoneDefectViewController : UIViewController <SBCollectionControllerDelegate>

// declare properties
@property (nonatomic, strong) NSNumber *elementId;
@property (nonatomic, strong) SBCollectionController *collectionController;

// the currently selected defect id
@property (nonatomic, strong) Defect *currentDefect;

@end
```

```
//
//  SBPhonePageCategoryViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 26-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBPhoneDefectViewController.h"
#import "SBMaintenanceInterface.h"

@implementation SBPhoneDefectViewController

/*  Prepare the view from being loaded  */
- (void) viewDidLoad
{
    [super viewDidLoad];

    // set the view controller's title
    [self setTitle: NSLocalizedString(@"defect.controller.title", nil)];
}

/*  In this method tell the delegate to change its NavigationBar    */
- (void) viewWillAppear:(BOOL)animated
{
    // call super with the same method
    [super viewWillAppear: animated];

    // check whether the collection controller was created, when not recreate it
    if (self.collectionController == nil) {

        // create the fetch request for the fetched results controller, the predicate will only select items with a
            parent location id that is passed to this view controller by the location selection controller
        NSPredicate *defectPredicate = [NSPredicate predicateWithFormat: @"element.elementId == %d", [_elementId
            intValue]];
        NSSortDescriptor *defectSortDescriptor = [NSSortDescriptor sortDescriptorWithKey: @"defectId" ascending: YES
            ];

        // create the fetch request and add the sort descriptor and predicate
        NSFetchRequest *defectRequest = [[NSFetchRequest alloc] initWithEntityName: @"Defect"];
        [defectRequest setPredicate: defectPredicate];
        [defectRequest setSortDescriptors: [NSArray arrayWithObject: defectSortDescriptor]];

        // create the collection controller for the defects
        self.collectionController = [[SBCollectionController alloc] initWithFrame: CGRectZero fetchRequest:
            defectRequest];
        [self.collectionController setDelegate: self];

        // get the current selected location id
        id<SBMaintenanceInterface> controller = (id<SBMaintenanceInterface>)[self.navigationController delegate];
        self.currentDefect = (Defect *)[controller viewController: self proceedingValueForKey: @"defect"];
    }

    // when the collectionview is not added add it to the view
    if ([self.collectionController superview] == nil) {

        [self.collectionController setFrame: self.view.bounds];
        [self.view addSubview: self.collectionController];
    }
}

/*  Prepares for the next viewcontroller    */
- (void) prepareForSegue:(UIStoryboardSegue *) segue sender:(id) sender
{
    // prepare for the next view controller
    if ([[segue identifier] isEqualToString: @"userSelectedCategory:"]) {

        // commit the selected defect id to the navigation controller delegate
        id<SBMaintenanceInterface> delegate = (id<SBMaintenanceInterface>) [[self navigationController] delegate];
        [delegate viewController: self didChangeValue: [self.collectionController managedObjectForSelectedIndexPath]
            forProceedingKey: @"defect"];
    }
}

/*  When a low memory warning is received free up some memory   */
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];

    // when the view is not on screen prepare memory
    if ([self.view window] == nil) {

        // remove the view from its superview when needed
        if ([self.collectionController superview] != nil) {
            [self.collectionController removeFromSuperview];
        }
```

```
        // release the controller
        self.collectionController = nil;
    }
}

#pragma mark    -
#pragma mark    Class method

/*  When an item gets selected  */
- (void) collectionController:(SBCollectionController *)controller didSelectItemWithIdentifier:(NSNumber *)
    identifier
{
    [self performSegueWithIdentifier: @"userSelectedCategory:" sender: self];
}

/*  When an item gets deselected    */
- (void) collectionControllerDidDeselectCollectionItem:(SBCollectionController *)controller
{

}



@end
```

```
//
//  SBPhonePageDescriptionViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 27-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBMaintenanceInterface.h"

// defines the viewController dismissal method
typedef enum
{
    SBAfterKeyboardActionPush              =   0,
    SBAfterKeyboardActionPop               =   1,
    SBAfterKeyboardActionPopNewProceeding  =   2,
    SBAfterKeyboardActionNone              =   3
}SBAfterKeyboardAction;

@interface SBPhoneDescriptionViewController : UIViewController <UITextViewDelegate, UIAlertViewDelegate>

// the textview which holds the description
@property (nonatomic, strong) UITextView *descriptionTextView;
@property (nonatomic) SBAfterKeyboardAction controllerAction;
@property (nonatomic) BOOL keyboardIsVisible;

// the current length of the description input
@property (nonatomic) NSInteger currentDescriptionLength;

// method that respond to the UIKeyboard (dis)appearance
- (void) keyboardDidAppear:(NSNotification *) notification;
- (void) keyboardWillDisappear:(NSNotification *) notification;
- (void) keyboardDidDisappear:(NSNotification *) notification;
- (void) keyboardWillAppear:(NSNotification *) notification;

// IBAction methods for next and back buttons
- (IBAction) nextButtonTapped:(UIBarButtonItem *) sender;
- (void) popViewController;

@end
```

```
//
//  SBPhonePageDescriptionViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 27-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBPhoneDescriptionViewController.h"
#import "SBKeyboardCustomToolbar.h"
#import "SBMaintenanceInterface.h"
#import "SBPhoneLocationViewController.h"

@implementation SBPhoneDescriptionViewController

/*  Initializer for the description selection view, make sure we listen to some notifications needed    */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    self = [super initWithCoder: aDecoder];
    if (self) {

        // add notifications listeners for the keyboard to show or hide
        [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboardDidAppear:) name:
            UIKeyboardWillShowNotification object: nil];
        [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboardWillDisappear:) name:
            UIKeyboardWillHideNotification object: nil];
        [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboardDidDisappear:) name:
            UIKeyboardDidHideNotification object: nil];
        [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboardWillAppear:) name:
            UIKeyboardWillShowNotification object: nil];

        // set the action selector to default
        self.controllerAction = SBAfterKeyboardActionNone;
        self.keyboardIsVisible = NO;
    }

    return self;
}

/*  When removed from memory remove from the notification center    */
- (void) dealloc
{
    // remove the observer from the NotificationCenter
    [[NSNotificationCenter defaultCenter] removeObserver: self];
}

/*  Method gets called from the navigation controller when the keyboard should be hidden first before popping */
- (void) popViewController
{
    if (self.keyboardIsVisible) {
        // force the resign of first responder and set the action after the keyboard is dismissed
        [self.descriptionTextView resignFirstResponder];
        self.controllerAction = SBAfterKeyboardActionPop;
    }
    else {
        // pop the controller
        [self.navigationController popViewControllerAnimated: YES];
    }
}

/*  When view is loaded from memory do initial controller startup and prepare the main view */
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    // set the title of the view controller
    [self setTitle: NSLocalizedString(@"description.controller.title", nil)];
}

/*  In this method tell the delegate to change its NavigationBar    */
- (void) viewDidAppear:(BOOL) animated
{
    // call super with method
    [super viewDidAppear: animated];

    // make the view first responder when possible
    if ([self.descriptionTextView canBecomeFirstResponder] && ![self.descriptionTextView isFirstResponder]) {
        [self.descriptionTextView becomeFirstResponder];
    }
}

/*  Disable buttons when animating in the view  */
- (void) viewWillAppear:(BOOL)animated
{
    // create the textview when not already created
    if (self.descriptionTextView == nil) {
```

```
        // create the UITextView
        self.descriptionTextView = [[UITextView alloc] initWithFrame: CGRectZero];

        // create the keyboard toolbar and add it to the keyboard
        SBKeyboardCustomToolbar *keyboardToolbar = [[SBKeyboardCustomToolbar alloc] initWithButtonTarget: self
            withSelector: @selector(nextButtonTapped:) frame: CGRectMake(0, 0, self.view.frame.size.width, 44)];
        [self.descriptionTextView setInputAccessoryView: keyboardToolbar];

        // configure the textview
        [self.descriptionTextView setKeyboardAppearance: UIKeyboardAppearanceDefault];
        [self.descriptionTextView setKeyboardType: UIKeyboardTypeAlphabet];
        [self.descriptionTextView setDelegate: self];
        [self.descriptionTextView setFont: [UIFont fontWithName: @"AppleGothic" size: 18]];

        // get the navigation controller delegate to retrieve the current description for the maintenance when
            available
        id<SBMaintenanceInterface> delegate = (id<SBMaintenanceInterface>) [self.navigationController delegate];
        NSString *existingDescription = [delegate viewController: self proceedingValueForKey:
            @"proceedingDescription"];

        // when the existing description is avaiable prefill the textview
        if (existingDescription != nil) {
            [self.descriptionTextView setText: existingDescription];
        }
    }

    // disable the left and right bar button items, this because when the user quickly presses the next button while
        an keyboard animation is busy, it can result in a corrupted navigation bar
    [[self.navigationItem leftBarButtonItem] setEnabled: NO];
    [[self.navigationItem rightBarButtonItem] setEnabled: NO];

    // only add when the textview is not already on screen
    if ([self.descriptionTextView superview] == nil) {

        // set bounds / frame of the description view, the correct view dimensions are now known
        [self.descriptionTextView setFrame: self.view.bounds];
        [[self view] addSubview: self.descriptionTextView];
    }
}

/*  When a memory warning is received clean up resources    */
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];

    // when the view is not on screen remove resources from memory
    if ([self.view window] == nil) {

        // when the description textview is part of a superview remove it from that view
        if ([self.descriptionTextView superview] == nil) {
            [self.descriptionTextView removeFromSuperview];
        }

        // release the textview
        self.descriptionTextView = nil;
    }
}

/*  Prepare for the next segue, this next controller will be the data selection controller  */
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // when the user has typed the description and wants to continue to the next view commit the description
    if ([[segue identifier] isEqualToString: @"userTypedDescription:"]) {

        // get the delegate navigation object and notify that object that the description value has changed
        id<SBMaintenanceInterface> delegate = (id<SBMaintenanceInterface>) [self.navigationController delegate];
        [delegate viewController: self didChangeValue: [self.descriptionTextView text] forProceedingKey:
            @"proceedingDescription"];
    }
}

#pragma mark -
#pragma mark Class methods

/*  When the keyboard will show on the screen, this method will be called   */
- (void) keyboardDidAppear:(NSNotification *) notification
{
    // enable the left and right bar button items, this because when the user quickly presses the next button while
        an keyboard animation is busy, it can result in a corrupted navigation bar
    [[self.navigationItem leftBarButtonItem] setEnabled: YES];
    [[self.navigationItem rightBarButtonItem] setEnabled: YES];

    // retreive the NSValue of the keyboard rectangle
    CGRect frameValue = [[[notification userInfo] valueForKey: UIKeyboardFrameEndUserInfoKey] CGRectValue];

    // calculate the new bounds fr the UITextView
```

```
    CGFloat viewHeight = [[UIScreen mainScreen] bounds].size.height - ([[UIApplication sharedApplication]
        statusBarFrame].size.height + [[self navigationController] navigationBar].frame.size.height + frameValue.
        size.height);

    // append the UITextView size
    [self.descriptionTextView setFrame: CGRectMake(0, 0, self.view.frame.size.width, viewHeight)];
}

/*  When the keyboard will disappear from the screen, this method will be called    */
- (void) keyboardWillDisappear:(NSNotification *) notification
{
    // disable the left and right bar button items, this because when the user quickly presses the next button while
        an keyboard animation is busy, it can result in a corrupted navigation bar
    [[self.navigationItem leftBarButtonItem] setEnabled: NO];
    [[self.navigationItem rightBarButtonItem] setEnabled: NO];
}

/*  when the keyboard appearance has finished   */
- (void) keyboardWillAppear:(NSNotification *)notification
{
    // set the keyboard to visible
    self.keyboardIsVisible = YES;
}

/*  When the keyboard did hide perform the next segue   */
- (void) keyboardDidDisappear:(NSNotification *)notification
{
    // set the keyboard flag to no
    self.keyboardIsVisible = NO;

    // disable the left and right bar button items, this because when the user quickly presses the next button while
        an keyboard animation is busy, it can result in a corrupted navigation bar
    [[self.navigationItem leftBarButtonItem] setEnabled: YES];
    [[self.navigationItem rightBarButtonItem] setEnabled: YES];

    // switch through the different types of actions that can be performed after keyboard dismissal
    switch (self.controllerAction)
    {
        // when the action is default, do nothing
        case SBAfterKeyboardActionNone:
            break;

        // pop to the previous view controller
        case SBAfterKeyboardActionPop:
            [[self navigationController] popViewControllerAnimated: YES];
            break;

        // push to the next view controller / segue
        case SBAfterKeyboardActionPush:
            [self performSegueWithIdentifier: @"userTypedDescription:" sender: self];
            break;

        case SBAfterKeyboardActionPopNewProceeding:
        {
            // loop through the view controller array of the navigation controller
            for (UIViewController *viewController in [self.navigationController viewControllers]) {
                if ([viewController isKindOfClass: [SBPhoneLocationViewController class]]) {
                    // create a new proceeding object after commiting the current description
                    id<SBMaintenanceInterface> delegate = (id<SBMaintenanceInterface>)[self.navigationController
                        delegate];
                    [delegate viewController: self didChangeValue: [self.descriptionTextView text] forProceedingKey:
                        @"proceedingDescription"];
                    [delegate viewControllerWillCreateNewProceeding: self];

                    // pop to the location selection view controller and reset this controller for the new
                        proceeding
                    SBPhoneLocationViewController *locationController = (SBPhoneLocationViewController *)
                        viewController;
                    [locationController reset];

                    // pop to location controller
                    [self.navigationController popToViewController: viewController animated: YES];
                }
            }
            break;
        }

        default:
            break;
    }
}

/*  Perform the next segue when the next button is tapped   */
- (IBAction) nextButtonTapped:(UIBarButtonItem *)sender
{
    // show the date controller after the keyboard is hidden
    self.controllerAction = SBAfterKeyboardActionPush;
```

```
    [self.descriptionTextView resignFirstResponder];

    // Uncomment the following to enable multiple defect uploads in one single reservation

    // show a alert view with the option to add another proceeding
    /*UIAlertView *newProceedingChoice = [[UIAlertView alloc] initWithTitle:
        NSLocalizedString(@"description.controller.continue.choice.title", nil) message:
        NSLocalizedString(@"description.controller.continue.choice.description", nil) delegate: self
        cancelButtonTitle: NSLocalizedString(@"description.controller.continue.new.title", nil) otherButtonTitles:
        NSLocalizedString(@"description.controller.continue.finish.title", nil), nil];

    // show the alertview
    [newProceedingChoice show];*/
}

#pragma mark    -
#pragma mark    UIAlertView delegate methods

/*
 ** Uncomment this code to enable multiple defect uploades
- (void) alertView:(UIAlertView *)alertView willDismissWithButtonIndex:(NSInteger)buttonIndex
{
    // the cancel button index corresponds to adding a new proceeding object
    if ([alertView cancelButtonIndex] == buttonIndex) {
        self.controllerAction = SBAfterKeyboardActionPopNewProceeding;
    }
    else {
        self.controllerAction = SBAfterKeyboardActionPush;
    }

    // resign first responder to activate the view transaction
    [self.descriptionTextView resignFirstResponder];
}*/

#pragma mark    -
#pragma mark    UITextViewDelegate

/*  Changes the current character count */
- (void) textViewDidChange:(UITextView *)textView
{
    // set the new length
    self.currentDescriptionLength = [[textView text] length];

    // get the keyboard toolbar view
    SBKeyboardCustomToolbar *toolbar = (SBKeyboardCustomToolbar *)[self.descriptionTextView inputAccessoryView];

    // change the text color to red when the text length is bigger than allowed
    if (self.currentDescriptionLength >= 256) {

        // set the character counter label to red colored
        [[toolbar characterCountLabel] setTextColor: [UIColor redColor]];
        [toolbar setShouldRedraw: YES];
    }
    else {
        // redraw the count label
        [toolbar redraw];
    }

    // set the current character counter text
    [[toolbar characterCountLabel] setText: [NSString stringWithFormat: @"%d / 256", self.currentDescriptionLength]]
        ;
}

/*  This method checks for the new length of the string not to be longer than the max allowed characters    */
- (BOOL) textView:(UITextView *)textView shouldChangeTextInRange:(NSRange)range replacementText:(NSString *)text
{
    // when the length of the characters will be larger disallow insertion
    if (range.length == 0 && (self.currentDescriptionLength + [text length]) > 256) {
        return NO;
    }
    else {
        return YES;
    }
}

@end
```

```
//
//  SBPhoneElementViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 04-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBPhoneDefectViewController.h"
#import "SBCollectionController.h"
#import "Element.h"

@interface SBPhoneElementViewController : UIViewController <SBCollectionControllerDelegate>

// the controller that holds the elements
@property (nonatomic, strong) NSNumber *spaceId;
@property (nonatomic, strong) SBCollectionController *collectionController;

// the current selected id of the element
@property (nonatomic, strong) Element *currentElement;

@end
```

```
//
//  SBPhoneElementViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 04-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBPhoneElementViewController.h"
#import "SBMaintenanceInterface.h"

@implementation SBPhoneElementViewController

/*  Initialize the view contents when loaded from memory    */
- (void)viewDidLoad
{
    [super viewDidLoad];

    // set the title of the view controller
    [self setTitle: NSLocalizedString(@"element.controller.title", nil)];
}

/*  Respond to low memory warnings to create some more free memory  */
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];

    // when the view is not on screen prepare memory
    if ([self.view window] == nil) {

        // remove the view from its superview when needed
        if ([self.collectionController superview] != nil) {
            [self.collectionController removeFromSuperview];
        }

        // release the controller
        self.collectionController = nil;
    }
}

/*  In this method tell the delegate to change its NavigationBar    */
- (void) viewWillAppear:(BOOL)animated
{
    // call super with the same method
    [super viewWillAppear: animated];

    // when the collection controller is not yet created do that now
    if (self.collectionController == nil) {

        // create the fetch request for the fetched results controller, the predicate will only select items with a
            parent location id that is passed to this view controller by the location selection controller
        NSPredicate *elementPredicate = [NSPredicate predicateWithFormat: @"space.spaceId == %d", [_spaceId intValue
            ]];
        NSSortDescriptor *elementSortDescriptor = [NSSortDescriptor sortDescriptorWithKey: @"elementId" ascending:
            YES];

        // create the fetch request and add the sort descriptor and predicate
        NSFetchRequest *elementRequest = [[NSFetchRequest alloc] initWithEntityName: @"Element"];
        [elementRequest setPredicate: elementPredicate];
        [elementRequest setSortDescriptors: [NSArray arrayWithObject: elementSortDescriptor]];

        // create the collection controller that holds the selection of elements
        self.collectionController = [[SBCollectionController alloc] initWithFrame: CGRectZero fetchRequest:
            elementRequest];
        [self.collectionController setDelegate: self];

        // get the current selected location id
        id<SBMaintenanceInterface> controller = (id<SBMaintenanceInterface>)[self.navigationController delegate];
        self.currentElement = (Element *)[controller viewController: self proceedingValueForKey: @"element"];
    }

    // add the collection controller to the main view
    if ([self.collectionController superview] == nil) {

        [self.collectionController setFrame: self.view.bounds];
        [self.view addSubview: self.collectionController];
    }
}

/*  Prepare for the next ViewController */
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // when the next segue is the space set the element id
    if ([[segue identifier] isEqualToString: @"userSelectedElement:"])
    {
        // cast the next controller and set the space id
        SBPhoneDefectViewController *controller = (SBPhoneDefectViewController *) [segue destinationViewController];
```

```
        [controller setElementId: [self.collectionController identifierForSelectedIndexPath]];

        // commit the selected element id
        id<SBMaintenanceInterface> delegate = (id<SBMaintenanceInterface>) [[self navigationController] delegate];
        [delegate viewController: self didChangeValue: [self.collectionController managedObjectForSelectedIndexPath]
            forProceedingKey: @"element"];
    }
}

#pragma mark    -

/*  When an item gets selected  */
- (void) collectionController:(SBCollectionController *)controller didSelectItemWithIdentifier:(NSNumber *)
    identifier
{
    [self performSegueWithIdentifier: @"userSelectedElement:" sender: self];
}

/*  When an item gets deselected    */
- (void) collectionControllerDidDeselectCollectionItem:(SBCollectionController *)controller
{

}


@end
```

```
//
//  SBPhonePageImageViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 27-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBDefectImageDescriptor.h"

@class SBImageCompatabilityView;

@interface SBPhoneImageViewController : UIViewController <UINavigationControllerDelegate,
    UIImagePickerControllerDelegate, UIActionSheetDelegate>

// the image URL of the image that is used for this defect, saved on disk
@property (nonatomic, strong) SBDefectImageDescriptor *imageDescriptor;
@property (nonatomic, strong) SBImageCompatabilityView *compatabilityView;

// the button for image selection
@property (nonatomic, strong) UIButton *imageSelectionButton;
@property (nonatomic) BOOL isObserving;
@property (nonatomic) BOOL isAbleToDraw;

// property indicates whether the view is initially loaded
@property (nonatomic) BOOL isViewInitiallyLoaded;

- (IBAction) nextButtonTapped:(UIBarButtonItem *) sender;
- (IBAction) photoSelectionButtonTapped:(UIButton *) sender;

@end
```

```
//
//  SBPhonePageImageViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 27-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBPhoneImageViewController.h"
#import <MobileCoreServices/MobileCoreServices.h>
#import <AssetsLibrary/AssetsLibrary.h>
#import <QuartzCore/QuartzCore.h>
#import "SBErrorView.h"
#import "SBPhoneLoginViewController.h"

// factory for compatability
#import "SBViewCompatabilityFactory.h"
#import "SBImageCompatabilityView.h"

@interface SBPhoneImageViewController()

// remove all subviews from the view controller's view
- (void) removeAllSubviews;

// place the image selection button in the given rect
- (void) showImageSelectionOptionsButton;

// draws the image descriptor and thumbnail in the view
- (void) drawExistingImage;

@end

@implementation SBPhoneImageViewController

/*  Initialize the image controller view depending on the OS    */
- (id) initWithCoder:(NSCoder *)aDecoder
{
    self = [super initWithCoder: aDecoder];

    if (self) {
        // load the compatability view
        self.compatabilityView = [SBViewCompatabilityFactory loadCompatableImageController];
    }

    return self;
}

/*  When the view is loaded from memory do som initial setup for the image view */
- (void)viewDidLoad
{
    [super viewDidLoad];
    // do any additional setup after loading the view.

    // set the title of the image view controller
    [self setTitle: NSLocalizedString(@"image.controller.title", nil)];

    UIBarButtonItem *rightButton = [[UIBarButtonItem alloc] initWithTitle:
        NSLocalizedString(@"image.controller.navigation.next.title", nil) style: UIBarButtonItemStylePlain target:
        self action: @selector(nextButtonTapped:)];

    // register the back bar button to a custom method
    [self.navigationItem setRightBarButtonItem: rightButton];

    // retreive the navigation controller delegate
    SBPhoneLoginViewController <SBMaintenanceInterface>*viewController = (SBPhoneLoginViewController <
        SBMaintenanceInterface>*)[[self navigationController] delegate];

    // get the already created image descriptor when available
    self.imageDescriptor = (SBDefectImageDescriptor *)[viewController viewController: self proceedingValueForKey:
        @"imageDescriptor"];

    // the view is loaded
    self.isViewInitiallyLoaded = YES;
}

/*  In this method tell the delegate to change its NavigationBar    */
- (void) viewWillAppear:(BOOL)animated
{
    // call super with method
    [super viewDidAppear: animated];

    // when no image is set show the image button
    if (self.imageDescriptor == nil) {

        // show button and actionsheet when first on this screen
        [self showImageSelectionOptionsButton];
    }
```

```
    else if (self.imageDescriptor != nil){

        // an existing image is available, set the observer
        if (!self.isObserving) {

            self.isObserving = YES;
            [self.imageDescriptor addObserver: self forKeyPath: @"isProcessing" options:
                NSKeyValueObservingOptionNew context: NULL];
        }

        // when the image is done processing that means the thumbnail is available
        if (![self.imageDescriptor isProcessing])
        {
            // remove all subviews
            [self removeAllSubviews];

            // draw the existing image on screen with the edit button
            [self drawExistingImage];
            [self showImageSelectionOptionsButton];
        }
        else
        {
            // show a loading indicator
            SBErrorView *loadingView = [[SBErrorView alloc] initWithFrame: self.view.bounds errorMessage:
                NSLocalizedString(@"image.controller.image.processing.description", nil) withImageType:
                SBErrorImageTypeLoading];

            // remove all subviews and add the new view
            [self removeAllSubviews];
            [self.view addSubview: loadingView];
        }
    }
}

/*  When the view is being presented execute this code  */
- (void) viewDidAppear:(BOOL)animated
{
    // show the image selector sheet when no image has been selected
    if (self.imageDescriptor == nil && self.isViewInitiallyLoaded) {

        // show the image selection button
        [self photoSelectionButtonTapped: nil];
        self.isViewInitiallyLoaded = NO;
    }

    // call super
    [super viewDidAppear: animated];
}

/*  Places the selection button in the provided CGRect  */
- (void) showImageSelectionOptionsButton
{
    // draw the button on the screen in compatability view
    [self.compatabilityView drawImageSelectionButtonForController: self];
}

/*  When received a memory warning remove all subviews from this view   */
- (void) didReceiveMemoryWarning
{
    // send message to the super of this controller
    [super didReceiveMemoryWarning];

    // check whether the view is on screen, when so do nothing else free up memory
    if ([self.view window] == nil) {

        // loop through all the subviews and remove them from the superview, these views will be added when needed
            in the viewWillAppear method
        for (UIView *subview in [self.view subviews]) {
            [subview removeFromSuperview];
        }

        // remove the observer
        if (self.imageDescriptor != nil && self.isObserving) {

            // unset observer
            self.isObserving = NO;
            [self.imageDescriptor removeObserver: self forKeyPath: @"isProcessing" context: NULL];
        }
    }
}

/*  Prepare for the next segue  */
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // send message to delegate when proceeding to the next step
    if ([[segue identifier] isEqualToString: @"userProducedPhoto:"]) {
```

```
        // message the navigation controller delegate with the new image
        id<SBMaintenanceInterface> controller = (id<SBMaintenanceInterface>) [self.navigationController delegate];
        [controller viewController: self didChangeValue: self.imageDescriptor forProceedingKey: @"imageDescriptor"];
    }
}

/*  when the view will disappear from screen save the image descriptor  */
- (void) viewWillDisappear:(BOOL)animated
{
    // unset as observer of the image descriptor when needed
    if (self.imageDescriptor != nil && self.isObserving){

        self.isObserving = NO;
        [self.imageDescriptor removeObserver: self forKeyPath: @"isProcessing" context: NULL];
    }

    // remove the button from the superview for redrawing to correct place
    if ([self.imageSelectionButton superview] != nil) {
        [self.imageSelectionButton removeFromSuperview];
    }
}

/*  When the user taps the button on the navigation bar to go to the next controller this method gets called    */
- (IBAction) nextButtonTapped:(UIBarButtonItem *)sender
{
    // perform the segue to the next controller
    [self performSegueWithIdentifier: @"userProducedPhoto:" sender: self];
}

/*  Removes all subviews    */
- (void) removeAllSubviews
{
    // create array that holds all constraints that must be remove
    NSMutableArray *constraintsToRemove = [[NSMutableArray alloc] init];

    // get all constraints from the view that need to be removed, this is necessary for iOS6 because when this does
        not happen the application will crash with NSGenericException. Does not happen with iOS 7
    for (NSLayoutConstraint *constraint in self.view.constraints) {
        if( [self.view.subviews containsObject: constraint.firstItem] || [self.view.subviews containsObject:
            constraint.secondItem] ) {
            [constraintsToRemove addObject: constraint];
        }
    }

    // remove the constraints from the view
    [self.view removeConstraints: constraintsToRemove];

    // for each subview in the parent view remove it
    for (UIView *subview in [self.view subviews]) {
        [subview removeFromSuperview];
    }
}

/*  Draws the image on the screen with a shadow border  */
- (void) drawExistingImage
{
    // draw the image using the corresponding capatability class
    [self.compatabilityView drawImageInControllerView: self];
    [self.compatabilityView drawImageSelectionButtonForController: self];
}

/*  When the photo selection button is tapped this method gets executed */
- (IBAction) photoSelectionButtonTapped:(UIButton *)sender
{
    // create the action sheet
    UIActionSheet *actionSheet = nil;

    if ([UIImagePickerController isSourceTypeAvailable: UIImagePickerControllerSourceTypeCamera]) {

        // show options for when the photo camera is available
        actionSheet = [[UIActionSheet alloc] initWithTitle: NSLocalizedString(@"", nil) delegate: self
            cancelButtonTitle: NSLocalizedString(@"actionsheet.button.image.cancel.title", nil)
            destructiveButtonTitle: nil otherButtonTitles: NSLocalizedString(@"actionsheet.button.image.new.title",
            nil), NSLocalizedString(@"actionsheet.button.image.cameraroll.title", nil), nil];
    } else if ([UIImagePickerController isSourceTypeAvailable: UIImagePickerControllerSourceTypePhotoLibrary]) {

        // show options for when only the photo library is available
        actionSheet = [[UIActionSheet alloc] initWithTitle: NSLocalizedString(@"", nil) delegate: self
            cancelButtonTitle: NSLocalizedString(@"actionsheet.button.image.cancel.title", nil)
            destructiveButtonTitle: nil otherButtonTitles:
            NSLocalizedString(@"actionsheet.button.image.cameraroll.title", nil), nil];
    }

    // when the actionsheet was created show it, else show an error message. The camera or other input source is
        currently not available
    if (actionSheet == nil) {
```

```
        // rare case that not a camera or album is available, show error message
        UIAlertView *alertview = [[UIAlertView alloc] initWithTitle:
            NSLocalizedString(@"image.controller.image.sources.unavailable.title", nil) message:
            NSLocalizedString(@"image.controller.image.sources.unavailable.description", nil) delegate: nil
            cancelButtonTitle: nil otherButtonTitles: NSLocalizedString(@"alertview.button.title.dismiss", nil), nil
            ];

        // show the alert
        [alertview show];

    } else {

        // show the actionsheet in the current view
        [actionSheet showInView: self.view];
    }
}

#pragma mark    -
#pragma mark    UIImagePickerController delegate methods

/*  When the user has finished picking the image he/she wants to use    */
- (void) imagePickerController:(UIImagePickerController *)picker didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    // check for an already existing image descriptori
    if (self.imageDescriptor != nil){
        // remove the image descriptor and backing file image
        [self.imageDescriptor deleteCurrentImage];

        // remove the observer and release the object
        if (self.isObserving) {
            [self.imageDescriptor removeObserver: self forKeyPath: @"isProcessing"];
            self.isObserving = NO;
        }
        self.imageDescriptor = nil;
    }

    // when the metadata key is set, a new image is taken with the camera
    if ([info objectForKey: UIImagePickerControllerMediaMetadata])
    {
        // save the image to disk when possible
        self.imageDescriptor = [[SBDefectImageDescriptor alloc] initWithImage: [info valueForKey:
            UIImagePickerControllerOriginalImage] metadata: [info valueForKey: UIImagePickerControllerMediaMetadata]
            ];
    }
    else if ([info objectForKey: UIImagePickerControllerReferenceURL])
    {
        // an image is picked with from the camera roll / photo library
        self.imageDescriptor = [[SBDefectImageDescriptor alloc] initWithReferenceURL: [info valueForKey:
            UIImagePickerControllerReferenceURL] metaData: nil];
    }
    else
    {
        // show a loading indicator
        SBErrorView *loadingView = [[SBErrorView alloc] initWithFrame: self.view.bounds errorMessage:
            NSLocalizedString(@"image.controller.image.processing.description", nil) withImageType:
            SBErrorImageTypeFailed];

        // remove all subviews and add the new view
        [self removeAllSubviews];
        [self.view addSubview: loadingView];
    }

    // when the image descriptor is successfully created start the image processing and show an image activity
        indicator
    if (self.imageDescriptor != nil)
    {
        // observer the isProcessing value of the imagedescriptor
        if (!self.isObserving) {
            [self.imageDescriptor addObserver: self forKeyPath: @"isProcessing" options:
                NSKeyValueObservingOptionNew context: NULL];
            self.isObserving = YES;
        }
        // show a loading indicator
        SBErrorView *loadingView = [[SBErrorView alloc] initWithFrame: self.view.bounds errorMessage:
            NSLocalizedString(@"image.controller.image.processing.description", nil) withImageType:
            SBErrorImageTypeLoading];

        // remove all subviews and add the new view
        [self removeAllSubviews];
        [self.view addSubview: loadingView];

        // start processing
        [self.imageDescriptor startProcessing];
    }

    // the image can not be drawn on screen yet when the view controller is being dismissed, so make sure this only
        happens when the controller is already dismissed
```

```
    self.isAbleToDraw = NO;

    // dismiss the image picker controller
    [self dismissViewControllerAnimated: YES completion:^
    {
        // set thay the controller is able to draw
        self.isAbleToDraw = YES;

        // check wether the image is already created, when so draw the image on screen
        if (![self.imageDescriptor isProcessing]) {
            [self drawExistingImage];
            [self showImageSelectionOptionsButton];
        }
    }];
}

/*  When the user cancels picking the image */
- (void) imagePickerControllerDidCancel:(UIImagePickerController *)picker
{
    // dismiss the image picker controller
    [self dismissViewControllerAnimated: YES completion: nil];
}

#pragma mark    -
#pragma mark    UIActionSheet delegate methods

/*  Gets called when the actionsheet is dismissed with a certain button press   */
- (void) actionSheet:(UIActionSheet *)actionSheet didDismissWithButtonIndex:(NSInteger)buttonIndex
{
    // when the cancel button (do not add image) has been tapped show next view
    if ([actionSheet cancelButtonIndex] == buttonIndex) {

        // show the next view controller
        [self performSegueWithIdentifier: @"userProducedPhoto:" sender: self];
        return;
    } else {

        // create the image picker controller
        UIImagePickerController *imagePickerController = nil;

        // when the camera is available process the correct button index
        if ([UIImagePickerController isSourceTypeAvailable: UIImagePickerControllerSourceTypeCamera]) {

            // create and configure the controller
            imagePickerController = [[UIImagePickerController alloc] init];
            [imagePickerController setMediaTypes: [NSArray arrayWithObject: (NSString *) kUTTypeImage]];

            // set the corresponding source type
            if (buttonIndex == 0) {
                [imagePickerController setSourceType: UIImagePickerControllerSourceTypeCamera];
            } else {
                [imagePickerController setSourceType: UIImagePickerControllerSourceTypePhotoLibrary];
            }

        } else if ([UIImagePickerController isSourceTypeAvailable: UIImagePickerControllerSourceTypePhotoLibrary]) {

            // create and configure the controller
            imagePickerController = [[UIImagePickerController alloc] init];
            [imagePickerController setMediaTypes: [NSArray arrayWithObject: (NSString *) kUTTypeImage]];
            [imagePickerController setSourceType: UIImagePickerControllerSourceTypePhotoLibrary];
        }

        // when the picker controller was created show it in the view
        if (imagePickerController != nil) {

            // set the delegate object for messages and show the view controller
            [imagePickerController setDelegate: self];
            [self presentViewController: imagePickerController animated: YES completion: nil];
        } else {

            // rare case that not a camera or album is available, show error message
            UIAlertView *alertview = [[UIAlertView alloc] initWithTitle:
                NSLocalizedString(@"image.controller.image.sources.unavailable.title", nil) message:
                NSLocalizedString(@"image.controller.image.sources.unavailable.description", nil) delegate: nil
                cancelButtonTitle: nil otherButtonTitles: NSLocalizedString(@"alertview.button.title.dismiss", nil),
                nil];

            // show the alert
            [alertview show];
        }
    }
}

#pragma mark    -
#pragma mark    Key value observing

/*  when the image descriptor changes the isprocessing value this method gets called    */
```

```
- (void) observeValueForKeyPath:(NSString *) keyPath ofObject:(id) object change:(NSDictionary *) change context:
    (void *) context
{
    // when the key path is the one we want
    if ([keyPath isEqualToString: @"isProcessing"])
    {
        // get the new value
        BOOL newValue = [[change valueForKey: NSKeyValueChangeNewKey] boolValue];

        // when newValue is NO the image has been processed
        if (!newValue)
        {
            // when the image descriptor encountered an error show the error view
            if ([self.imageDescriptor processingError] != nil)
            {
                // remove all subviews
                [self removeAllSubviews];
            }
            else
            {
                // remove all subviews
                [self removeAllSubviews];

                // draw the received on the screen and the button to edit this image when available
                if (self.isAbleToDraw) {
                    [self drawExistingImage];
                    [self showImageSelectionOptionsButton];
                }
            }

        }
    }
}

@end
```

```
//
//  SBPhoneLocationViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 12-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBPhoneSpaceViewController.h"
#import "SBCollectionController.h"
#import "Location.h"

@class SBLocationCompatabilityView;

@interface SBPhoneLocationViewController : UIViewController <SBCollectionControllerDelegate>

// the controller with the collection view objects
@property (nonatomic, strong) SBCollectionController *collectionController;

// current selected location id
@property (nonatomic, strong) Location *currentLocation;

// cancels the creation of this maintenance object
- (IBAction) cancelButtonTapped:(UIBarButtonItem *) sender;

// prepare the controller for a new proceeding
- (void) reset;

@end
```

```
//
//  SBPhoneLocationViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 12-07-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBPhoneLocationViewController.h"
#import "SBMaintenanceInterface.h"
#import "SBLocationCompatabilityView.h"
#import "SBViewCompatabilityFactory.h"

@implementation SBPhoneLocationViewController

/*  When the view is first loaded prepare the view contents */
- (void) viewDidLoad
{
    [super viewDidLoad];

    // Do any additional setup after loading the view.
    [self setTitle: NSLocalizedString(@"location.controller.title", nil)];
}

/*  Remove any recreatable objects from memory in case of a low memory warning  */
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];

    // when the view is not on screen we can remove some objects from memory
    if ([self.view window] == nil) {

        // remove the collection controller from superview when needed
        if ([self.collectionController superview] != nil) {
            [self.collectionController removeFromSuperview];
        }

        // release the collection controller
        self.collectionController = nil;
    }
}

/*  When the location controller will appear prepare the view contents such as the collection controller    */
- (void) viewWillAppear:(BOOL)animated
{
    [super viewWillAppear: animated];

    // when the collection view controller was not yet instantiated create it
    if (self.collectionController == nil) {

        // create a fetch request to use with the fetched results controller
        NSSortDescriptor *locationSortDescriptor = [NSSortDescriptor sortDescriptorWithKey: @"locationId" ascending:
            YES];
        NSFetchRequest *request = [[NSFetchRequest alloc] initWithEntityName: @"Location"];
        [request setSortDescriptors: [NSArray arrayWithObject: locationSortDescriptor]];

        // create the collection controller for all locations
        self.collectionController = [[SBCollectionController alloc] initWithFrame: CGRectZero fetchRequest: request]
            ;
        [self.collectionController setDelegate: self];

        // get the current selected location id
        id<SBMaintenanceInterface> controller = (id<SBMaintenanceInterface>)[self.navigationController delegate];
        self.currentLocation = (Location *)[controller viewController: self proceedingValueForKey: @"location"];
    }

    // when the status of the view is none, start loading the location data
    if ([self.collectionController superview] == nil) {

        // set the frame and add to the main view
        [self.collectionController setFrame: self.view.bounds];
        [self.view addSubview: self.collectionController];
    }
}

/*  Reset the selected contents of the view controller  */
- (void) reset
{
    [self.collectionController reset];
}

#pragma mark    -
#pragma mark    Class methods

/*  Prepare the controller forthe view content */
- (void) prepareForSegue:(UIStoryboardSegue *) segue sender:(id) sender
{
```

```
    // when the segue identifier is equal to user selected an item
    if ([[segue identifier] isEqualToString: @"userSelectedLocation:"]) {
        // pass the selected location id to the next controller
        SBPhoneSpaceViewController *controller = (SBPhoneSpaceViewController *) [segue destinationViewController];
        [controller setLocationId: [self.collectionController identifierForSelectedIndexPath]];

        // commit the location id
        id<SBMaintenanceInterface> delegate = (id<SBMaintenanceInterface>) [self.navigationController delegate];
        [delegate viewController: self didChangeValue: [self.collectionController managedObjectForSelectedIndexPath]
            forProceedingKey: @"location"];
    }
}

/*  Gets notified by the parent that the next view controller should be shown   */
- (void) shouldProgressToNextViewControllerWithIdentifier:(NSNumber *)currentIdentifier
{
    // perform the segue to the next controller
    [self performSegueWithIdentifier: @"userSelectedLocation:" sender: self];
}

/*  When the cancel button has been pressed */
- (IBAction) cancelButtonTapped:(UIBarButtonItem *)sender
{
    // dismiss the modal view controller
    [self dismissViewControllerAnimated: YES completion: nil];
}

#pragma mark    -

/*  When an item gets selected  */
- (void) collectionController:(SBCollectionController *)controller didSelectItemWithIdentifier:(NSNumber *)
    identifier
{
    [self performSegueWithIdentifier: @"userSelectedLocation:" sender: self];
}

/*  When an item gets deselected    */
- (void) collectionControllerDidDeselectCollectionItem:(SBCollectionController *)controller
{

}

@end
```

```
//
//  SBPhoneLoginViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 25-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBMaintenanceInterface.h"
#import "SBTextField.h"
#import "SBPhoneCustomerViewController.h"
#import "SBCustomerControllerDelegate.h"
#import "SBKeyboardToolbarDelegate.h"

@class SBLoginCompatabilityView, SBUser;

@interface SBPhoneLoginViewController : UIViewController <SBOperationDelegate, SBURLOperationDelegate,
    UITableViewDataSource, UITableViewDelegate, UITextFieldDelegate, UINavigationControllerDelegate,
    UIAlertViewDelegate, SBCustomerControllerDelegate, SBKeyboardToolbarDelegate>

// declare all different textfields
@property (nonatomic, strong) SBTextField *firstnameField;
@property (nonatomic, strong) SBTextField *telephone1Field;
@property (nonatomic, strong) SBTextField *telephone2Field;
@property (nonatomic, strong) SBTextField *emailField;

// the user object for this reservation
@property (nonatomic, strong) SBUser *currentUser;

// address information textfields
@property (nonatomic, strong) SBTextField *zipcodeField;
@property (nonatomic, strong) SBTextField *houseNumberField;
@property (nonatomic, strong) SBTextField *streetField;
@property (nonatomic, strong) SBTextField *townField;

// to know what the previous and the current textfield objects are
@property (nonatomic, weak) SBTextField *currentTextField;
@property (nonatomic, weak) SBTextField *previousTextField;

// the current user of the application
@property (nonatomic, strong) SBLoginCompatabilityView *compatabilityView;

// the tableview to hold each uitextfield
@property (nonatomic, strong) UITableView *tableView;

// the UUID of the operation that is currently executing
@property (nonatomic, strong) NSUUID *addressOperationUUID;

// helper variables for keyboard showing and hiding transitions, this one is used to determine whether the next
    button on the keyboard has been pressed. This way we can tell whether to show the view controller automatic for
    customer selection
@property (nonatomic) BOOL isCustomerControllerLoadedViaKeyboardNext;

// whether the customer field can be marked as invalid
@property (nonatomic) BOOL canMarkCustomerSelectionFieldAsInvalid;

// methods for keyboard appearance and disappearance
- (void) keyboardDidAppear:(NSNotification *) notification;
- (void) keyboardWillDisappear:(NSNotification *) notification;

// IBAction methods that respond to the cancel or continue actions
- (IBAction) addMaintenanceButtonTapped:(UIBarButtonItem *) sender;
- (IBAction) cancelButtonTapped:(UIBarButtonItem *) sender;

@end
```

```
//
//  SBPhoneLoginViewController.m
//  ;
//
//  Created by Stephan de Bakker on 25-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBPhoneLoginViewController.h"
#import "SBOperationManager.h"
#import "SBUser.h"
#import "SBURLOperation.h"
#import "SBViewCompatabilityFactory.h"
#import "SBLoginCompatabilityView.h"
#import "SBPhoneDescriptionViewController.h"
#import "SBPhoneOverviewViewController.h"

#import "SBAPIRequest.h"
#import "SBAPIResponse.h"

// keyboard toolbar classes
#import "SBKeyboardToolbarLayout.h"
#import "SBKeyboardToolbar.h"

// identifier for the address lookup operation
static NSString *SBAPIAddressLookupOperationName    =   @"SBAPIAddressLookupOperationName";

// API result dictionary keys
static NSString * const SBZipCodeLookUpKeyCity                  =   @"city";
static NSString * const SBZipCodeLookUpKeyHouseNumber           =   @"house_number";
static NSString * const SBZipCodeLookUpKeyHouseNumberAddition   =   @"house_number_addition";
static NSString * const SBZipCodeLookUpKeyLatitude              =   @"latitude";
static NSString * const SBZipCodeLookUpKeyLongitude             =   @"longitude";
static NSString * const SBZipCodeLookUpKeyZipCode               =   @"zipcode";
static NSString * const SBZipCodeLookUpKeyStreet                =   @"street";

@interface SBPhoneLoginViewController ()

// gets called when text in the textfield changes
- (BOOL) textFieldDidChangeNotification:(NSNotification *) notification;

// when the user taps the tableview and no cell this method gets called and will dismiss the keyboard when necessary
- (void) tapGestureOnTableViewReceived:(UITapGestureRecognizer *) gesture;

// start the validating process of all textfields
- (BOOL) inputFieldsHaveValidContent;

@end

@implementation SBPhoneLoginViewController

#pragma mark View Lifecycle methods & Callbacks

/*  Initializer method for the login view controller    */
- (id) initWithCoder:(NSCoder *)aDecoder
{
    self = [super initWithCoder: aDecoder];

    if (self)
    {
        // create the user with no values
        self.compatabilityView = [SBViewCompatabilityFactory loadCompatableLoginController];

        self.isCustomerControllerLoadedViaKeyboardNext = NO;
        self.canMarkCustomerSelectionFieldAsInvalid = NO;
    }

    return self;
}

/*  When removed from memory remove from the notification center    */
- (void) dealloc
{
    // remove from notification center
    [[NSNotificationCenter defaultCenter] removeObserver: self];
}

/*  When the view gto loaded in memory prepare the user interface of the login view */
- (void) viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    // set the title of the login page
    [self setTitle: NSLocalizedString(@"login.controller.title", nil)];

    // Custom initialization, register for notification
```

```
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboardDidAppear:) name:
        UIKeyboardWillShowNotification object: nil];
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(keyboardWillDisappear:) name:
        UIKeyboardWillHideNotification object: nil];

    // draw iOS specific interface
    [self.compatabilityView initializeInterfaceOfViewController: self];

    // set the current user
    self.currentUser = [(id<SBMaintenanceInterface>)[self.navigationController delegate] viewController: self
        proceedingValueForKey: @"reservationUser"];

    // create the UITableView with a zero CGRect
    self.tableView = [[UITableView alloc] initWithFrame: CGRectZero style: UITableViewStyleGrouped];

    // create the toolbar for the keyboard toolbar
    SBKeyboardToolbar *keyboardToolbar = [[SBKeyboardToolbar alloc] initWithFrame: CGRectMake(0, 0, 320, 44)];
    [keyboardToolbar setDelegateObject: self];

    // add a tap gesture recognizer to the table view to allow the dismissal of the keyboard
    UITapGestureRecognizer *gestureRecognizer = [[UITapGestureRecognizer alloc] initWithTarget: self action:
        @selector(tapGestureOnTableViewReceived:)];
    [gestureRecognizer setCancelsTouchesInView: NO];
    [self.tableView addGestureRecognizer: gestureRecognizer];

    // set placeholders for the textfields
    [self.emailField setPlaceholder: NSLocalizedString(@"login.controller.textfield.email.placeholder", nil)];
    [self.firstnameField setPlaceholder: NSLocalizedString(@"login.controller.textfield.name.placeholder", nil)];
    [self.telephone1Field setPlaceholder: NSLocalizedString(@"login.controller.textfield.phone.placeholder", nil)];
    [self.telephone2Field setPlaceholder: NSLocalizedString(@"login.controller.textfield.phone.optional.placeholder"
        , nil)];
    [self.zipcodeField setPlaceholder: NSLocalizedString(@"login.controller.textfield.zipcode.placeholder", nil)];
    [self.houseNumberField setPlaceholder: NSLocalizedString(@"login.controller.textfield.housenumber.placeholder",
        nil)];

    // append the keyboard type for the email en telephone fields, do not use autocapatilization for the email field
    [self.emailField setKeyboardType: UIKeyboardTypeEmailAddress];
    [self.emailField setAutocapitalizationType: UITextAutocapitalizationTypeNone];
    [self.emailField setAutocorrectionType: UITextAutocorrectionTypeNo];
    [self.zipcodeField setKeyboardType: UIKeyboardTypeNamePhonePad];
    [self.houseNumberField setKeyboardType: UIKeyboardTypeNamePhonePad];

    // keyboard for numbers when editing phone numbers
    [self.telephone1Field setKeyboardType: UIKeyboardTypeNumbersAndPunctuation];
    [self.telephone2Field setKeyboardType: UIKeyboardTypeNumbersAndPunctuation];

    // set delegates for each UITextField object
    [self.emailField setDelegate: self];
    [self.firstnameField setDelegate: self];
    [self.telephone1Field setDelegate: self];
    [self.telephone2Field setDelegate: self];
    [self.zipcodeField setDelegate: self];
    [self.houseNumberField setDelegate: self];

    // set address fields street and town disabled
    [self.streetField setEnabled: NO];
    [self.townField setEnabled: NO];

    // set the corresponding indexpaths and scroll positions of the textfields
    [self.firstnameField setCorrespondingIndexPath: [NSIndexPath indexPathForRow: 0 inSection: 0]];
    [self.emailField setCorrespondingIndexPath: [NSIndexPath indexPathForRow: 1 inSection: 0]];
    [self.telephone1Field setCorrespondingIndexPath: [NSIndexPath indexPathForRow: 2 inSection: 0]];
    [self.telephone2Field setCorrespondingIndexPath: [NSIndexPath indexPathForRow: 3 inSection: 0]];
    [self.zipcodeField setCorrespondingIndexPath: [NSIndexPath indexPathForRow: 0 inSection: 1]];
    [self.houseNumberField setCorrespondingIndexPath: [NSIndexPath indexPathForRow: 0 inSection: 1]];

    // set scroll positions for the textfields in the tableview
    [self.firstnameField setTextfieldScrollPosition: UITableViewScrollPositionTop];
    [self.emailField setTextfieldScrollPosition: UITableViewScrollPositionMiddle];
    [self.telephone1Field setTextfieldScrollPosition: UITableViewScrollPositionMiddle];
    [self.telephone2Field setTextfieldScrollPosition: UITableViewScrollPositionMiddle];
    [self.zipcodeField setTextfieldScrollPosition: UITableViewScrollPositionTop];
    [self.houseNumberField setTextfieldScrollPosition: UITableViewScrollPositionTop];

    // configure the textfields to set the next textfield connection
    [self.firstnameField setNextTextField: self.emailField];
    [self.emailField setNextTextField: self.telephone1Field];
    [self.telephone1Field setNextTextField: self.telephone2Field];
    [self.zipcodeField setNextTextField: self.houseNumberField];
    [self.emailField setPreviousTextField: self.firstnameField];
    [self.telephone1Field setPreviousTextField: self.emailField];
    [self.telephone2Field setPreviousTextField: self.telephone1Field];
    [self.houseNumberField setPreviousTextField: self.zipcodeField];

    // configure the clear button, the x shown at the end of each textfield input to clear the field
    [self.emailField setClearButtonMode: UITextFieldViewModeWhileEditing];
    [self.firstnameField setClearButtonMode: UITextFieldViewModeWhileEditing];
```

```
    [self.telephone2Field setClearButtonMode: UITextFieldViewModeWhileEditing];
    [self.telephone1Field setClearButtonMode: UITextFieldViewModeWhileEditing];
    [self.zipcodeField setClearButtonMode: UITextFieldViewModeWhileEditing];
    [self.houseNumberField setClearButtonMode: UITextFieldViewModeWhileEditing];

    // set the keyboard toolbar for each textfield
    [self.emailField setInputAccessoryView: keyboardToolbar];
    [self.firstnameField setInputAccessoryView: keyboardToolbar];
    [self.telephone1Field setInputAccessoryView: keyboardToolbar];
    [self.telephone2Field setInputAccessoryView: keyboardToolbar];
    [self.zipcodeField setInputAccessoryView: keyboardToolbar];
    [self.houseNumberField setInputAccessoryView: keyboardToolbar];

    // set gray textcolors for the inaccesable fields
    [self.streetField setTextColor: [UIColor grayColor]];
    [self.townField setTextColor: [UIColor grayColor]];

    // set the toolbar description titles
    [self.firstnameField setToolbarInfo: NSLocalizedString(@"login.controller.keyboard.firstname.title", nil)];
    [self.emailField setToolbarInfo: NSLocalizedString(@"login.controller.keyboard.email.title", nil)];
    [self.telephone1Field setToolbarInfo: NSLocalizedString(@"login.controller.keyboard.phonenumber.title", nil)];
    [self.telephone2Field setToolbarInfo: NSLocalizedString(@"login.controller.keyboard.optional.phonenumber.title",
        nil)];
    [self.zipcodeField setToolbarInfo: NSLocalizedString(@"login.controller.keyboard.zipcode.title", nil)];
    [self.houseNumberField setToolbarInfo: NSLocalizedString(@"login.controller.keyboard.housenumber.title", nil)];

    // set tableview delegate and datasource
    [self.tableView setDelegate:self];
    [self.tableView setDataSource: self];

    // get notified when the zipcode and house number fields change its values
    // listen for text change notifications
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(textFieldDidChangeNotification:)
        name: UITextFieldTextDidChangeNotification object: self.zipcodeField];
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(textFieldDidChangeNotification:)
        name: UITextFieldTextDidChangeNotification object: self.houseNumberField];

    // try to fetch the user data from the userdefaults
    if ([self.currentUser fetchFromUserDefaults]) {

        // prefill the textfields
        [self.firstnameField setText: [self.currentUser userFirstName]];
        [self.emailField setText: [self.currentUser userEmail]];
        [self.telephone1Field setText: [self.currentUser userTelephoneNumber]];
        [self.telephone2Field setText: [self.currentUser userOptionalTelephoneNumber]];
        [self.zipcodeField setText: [self.currentUser zipcode]];
        [self.houseNumberField setText: [NSString stringWithFormat: @"%@", [self.currentUser houseNumber]]];
        [self.streetField setText: [self.currentUser street]];
        [self.townField setText: [self.currentUser town]];
    }
}

/*  when the tap gesture recognizer fires, the user wants to dismiss the tableview  */
- (void) tapGestureOnTableViewReceived:(UITapGestureRecognizer *) gesture
{
    // when there is a current textfield set, and this is the first responder and is able to resign being the first
        responder dismiss the keyboard
    if (self.currentTextField != nil && [self.currentTextField isFirstResponder] && [self.currentTextField
        canResignFirstResponder]) {

        // hide the current textfield being presented
        [self.currentTextField resignFirstResponder];
    }
}

/*  When view will be layed out, because we now know the correct dimensions we can set the tableview frame  */
- (void) viewWillLayoutSubviews
{
    // check for the tableview to be created
    if (self.tableView == nil) {

        // create the UITableView with a zero CGRect
        self.tableView = [[UITableView alloc] initWithFrame: CGRectZero style: UITableViewStyleGrouped];

        // reset the delegate and datasource
        [self.tableView setDelegate: self];
        [self.tableView setDataSource: self];
    }

    // when the tableview has no superview, add it to the view hierarchie
    if ([self.tableView superview] == nil) {

        // set bounds and add to the main view
        [self.tableView setFrame: self.view.bounds];
        [self.view addSubview: self.tableView];
    }
}
```

```
/*  Get rid of unused resources when a memory warning is received   */
- (void) didReceiveMemoryWarning
{
    // call super for further memory processing
    [super didReceiveMemoryWarning];

    // check whether this view is on screen, when on screen nothing can be free to make sure the view works
        correctly
    if ([self.view window] == nil) {

        // remove from the superview when memory is low
        if ([self.tableView superview] != nil) {
            [self.tableView removeFromSuperview];
        }

        // release the tableview
        self.tableView = nil;
    }
}

#pragma mark -
#pragma mark Defined Class methods

/*  Retrieve whether the inputted content is valid yes or no    */
- (BOOL) inputFieldsHaveValidContent
{
    // create the characterset string for the validation of the name values
    NSCharacterSet *allowedCharacters = [NSCharacterSet characterSetWithCharactersInString:
        @"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ "];
    BOOL invalidFieldFound = NO;

    // validate the inputted email address
    if ([SBUser isValidEmail: [self.emailField text]]) {

        // process the users email address to the user object
        [self.currentUser setUserEmail: [self.emailField text]];
    } else {

        // mark the email field as invalid input
        [self.emailField setIsMarkedAsInvalid: YES];
        invalidFieldFound = YES;
    }

    // validate the users name
    if ([SBUser validateString: [self.firstnameField text] withAllowedCharacterSet: allowedCharacters minLength: 2
        maxLength: 64]) {

        // set the first name to the user when correctly validated
        [self.currentUser setUserFirstName: [self.firstnameField text]];
    } else {

        // mark the firstname field as invalid
        [self.firstnameField setIsMarkedAsInvalid: YES];
        invalidFieldFound = YES;
    }

    // validate the first phone number
    if ([SBUser validatePhoneNumber: [self.telephone1Field text]]) {

        // process the telephone number to the user object
        [self.currentUser setUserTelephoneNumber: [self.telephone1Field text]];
    } else {

        // mark the first telephone as invalid
        [self.telephone1Field setIsMarkedAsInvalid: YES];
        invalidFieldFound = YES;
    }

    // validate the first phone number when set
    if ([[self.telephone2Field text] length] > 0) {

        if ([SBUser validatePhoneNumber: [self.telephone2Field text]]) {

            // process the telephone number to the user object
            [self.currentUser setUserTelephoneNumber: [self.telephone2Field text]];
        } else {

            // mark the first telephone as invalid
            [self.telephone2Field setIsMarkedAsInvalid: YES];
            invalidFieldFound = YES;
        }
    }

    // check whether a customer has been selected
    if ([self.currentUser userCustomer] == nil) {
        invalidFieldFound = YES;
```

```
        self.canMarkCustomerSelectionFieldAsInvalid = YES;
    }

    // validate whether the users address has been set
    if ([self.currentUser zipcode] == nil && [self.currentUser houseNumber] == nil) {

        // set invalid fields
        [self.zipcodeField setIsMarkedAsInvalid: YES];
        [self.houseNumberField setIsMarkedAsInvalid: YES];
        invalidFieldFound = YES;
    }

    // reload table view data
    [self.tableView reloadData];

    // return the validation result
    return invalidFieldFound;
}

/*  When the cancel button is tapped you can return to the main page    */
- (IBAction) cancelButtonTapped:(UIBarButtonItem *)sender
{
    // dismiss the view controller via the delegate object
    UIViewController *delegateController = (UIViewController *)[self.navigationController delegate];
    [delegateController dismissViewControllerAnimated: YES completion: nil];
}

/*  Responds to the add maintenance button  */
- (IBAction) addMaintenanceButtonTapped:(UIBarButtonItem *) sender
{
    // only continue when the user has validated the information
    if (![self inputFieldsHaveValidContent]) {

        // perform the segue to add a maintenance
        [self performSegueWithIdentifier: @"userWantsToAddMaintenance:" sender: self];

        // synchrinize user data
        [self.currentUser save];
    }
    else {

        // show alert view with localized titles and description
        UIAlertView *alertview = [[UIAlertView alloc] initWithTitle:
            NSLocalizedString(@"login.controller.validation.error.title", nil) message:
            NSLocalizedString(@"login.controller.validation.error.description", nil) delegate: nil
            cancelButtonTitle: NSLocalizedString(@"alertview.button.title.dismiss", nil) otherButtonTitles:  nil];
        [alertview show];
    }
}

/*  Prepare the next view controllers when needed   */
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // when the current textfield is available resign first responder
    if (self.currentTextField != nil){

        // hide first responder
        [self.currentTextField resignFirstResponder];
    }

    // when a new maintenance wizard will be shown add this class as a UINAvigationController delegate for callbacks
    if ([[segue identifier] isEqualToString: @"userWillSelectCustomer:"]) {

        // cast the root view controller as the customer view controller
        SBPhoneCustomerViewController *controller = (SBPhoneCustomerViewController *)[[segue
            destinationViewController] topViewController];

        // set the delegate of the object
        [controller setDelegate: self];
    }
}

#pragma mark -
#pragma mark    UINavigationController delegate methods

#pragma mark    UITableView delegate & datasource

/*  Returns the amount of cells in a given section of the tableview */
- (NSInteger) tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    // return the correct amount of cells for each section
    if (section == 0) {

        // the personal info section contains 5 cells
        return 5;
    }
    else {
```

```
        // the address section containes 3 cells
        return 3;
    }
}

/*  Returns the number of sections in this UITableView  */
- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView
{
    // this tableView holds 2 sections
    return 2;
}

/*  Returns a UITableViewCell for a specific IndexPath, each cell will hold a UITextField in this controller    */
- (UITableViewCell *) tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // create a cell, do not create from the dequeue
    static NSString *cellReuseIdentifier    =   @"userInformationCell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier: cellReuseIdentifier];

    // whether to draw the cell invalid border
    BOOL shouldDrawInvalidCellBorder = NO;

    // the custom subview textfield that should be added to the cell
    SBTextField *correspondingTextField = nil;

    // when no cell could be dequeued create a new one
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle: UITableViewCellStyleDefault reuseIdentifier:
            cellReuseIdentifier];
    }
    else {

        // when an existing cell was fetched reset the border settings
        cell.layer.borderWidth = 0.0f;

        // the cell will be reused, remove subviews
        UIView *contentView = [[cell subviews] objectAtIndex: 0];

        // vind the textfield and remove it from the cell
        for (UIView *subview in [contentView subviews]) {
            if ([subview isKindOfClass: [SBTextField class]]) {
                [subview removeFromSuperview];
                break;
            }
        }

        // reset the color, alignment and text of the cell
        [[cell textLabel] setFont: [UIFont boldSystemFontOfSize: [UIFont systemFontSize]]];
        [[cell textLabel] setTextColor: [UIColor blackColor]];
        [[cell textLabel] setTextAlignment: NSTextAlignmentCenter];
        [[cell textLabel] setText: @""];
    }

    // select the correct uitextfield view to add to the tableviewcell
    if ([indexPath section] == 0) {

        // show the firstname, suffix and lastname in the following three textfields
        if ([indexPath row] == 0){

            // set the corresponding textfield
            correspondingTextField = self.firstnameField;
        }
        else if ([indexPath row] == 1){

            // set the email field as the corresponding field
            correspondingTextField = self.emailField;
        }
        else if ([indexPath row] == 2){

            // set the first telephone number field as the corresponding field
            correspondingTextField = self.telephone1Field;
        }
        else if ([indexPath row] == 3){

            // set the optional telephone number field as the corresponding textField
            correspondingTextField = self.telephone2Field;
        }
        else if ([indexPath row] == 4) {

            // set the custom font of the cell
            [[cell textLabel] setFont: [UIFont boldSystemFontOfSize: 17.0f]];

            // check wether a customer is already selected by the user
            if ([self.currentUser userCustomer] != nil){
                [[cell textLabel] setText: [[self.currentUser userCustomer] customerName]];
            }
```

```
            else {
                // set info that a customer should be selected
                [[cell textLabel] setText: NSLocalizedString(@"login.controller.select.housing.title", nil)];
                shouldDrawInvalidCellBorder = self.canMarkCustomerSelectionFieldAsInvalid;
            }

            // text alignment will be centered for this cell
            [[cell textLabel] setTextAlignment: NSTextAlignmentCenter];
        }
    } else if ([indexPath section] == 1) {

        // show the zipcode and house number in the first cell
        if ([indexPath row] == 0) {

            // custom add multple cells to the subview
            [cell addSubview: self.zipcodeField];
            [cell addSubview: self.houseNumberField];

            // when the address is invalid draw the border
            if ([self.zipcodeField isMarkedAsInvalid]) {
                shouldDrawInvalidCellBorder = YES;
            }
        }
        else if ([indexPath row] == 1) {

            // add the street view to the cell
            correspondingTextField = self.streetField;
        }
        else if ([indexPath row] == 2) {

            // add the town identifier field as corresponding textField
            correspondingTextField = self.townField;
        }
    }

    // when the corresponding textfield is set add it to the cell subview
    if (correspondingTextField != nil) {

        // draw red border when the textfield has invalid content
        if ([correspondingTextField isMarkedAsInvalid]) {

            // set the color and format of the border of this cell
            cell.layer.borderColor = [UIColor redColor].CGColor;
            cell.layer.masksToBounds = YES;
            cell.layer.borderWidth = 1.0f;
            cell.layer.cornerRadius = 5.0f;
        }

        // add the textField to the cell
        [cell addSubview: correspondingTextField];
    } else if (shouldDrawInvalidCellBorder) {

        // set the color and format of the border of this cell
        cell.layer.borderColor = [UIColor redColor].CGColor;
        cell.layer.masksToBounds = YES;
        cell.layer.borderWidth = 1.0f;
        cell.layer.cornerRadius = 5.0f;
    }

    // cells cannot be selected exept for the indexpath 0, 4
    if ([indexPath section] != 0 && [indexPath row] != 4) {
        [cell setSelectionStyle: UITableViewCellSelectionStyleNone];
    }

    return cell;
}

/*  Returns a title for the header of a section */
- (NSString *) tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section
{
    // return the corresponding name for the section
    if (section == 0) {

        // return the title for the personal details section
        return NSLocalizedString(@"login.controller.tableview.section.personal.title", nil);
    }
    else {

        // return the title for the address information section
        return NSLocalizedString(@"login.controller.tableview.section.address.title", nil);
    }
}

/*  when a row is selected that could be selected this method is called */
- (void) tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    // remove the cell border for the customer
```

```
    self.canMarkCustomerSelectionFieldAsInvalid = NO;
    [tableView reloadRowsAtIndexPaths: [NSArray arrayWithObject: indexPath] withRowAnimation:
        UITableViewRowAnimationNone];

    // when the customer selection indexpath is selected continue to the next view controller for customer selection
    if ([indexPath section] == 0 && [indexPath row] == 4) {
        [self performSegueWithIdentifier: @"userWillSelectCustomer:" sender: self];
    }
}

#pragma mark -
#pragma mark    SBOperation delegate methods

/*  When the address lookup operation failed execution  */
- (void) operation:(SBOperation *)operation didFailWithError:(NSError *)error
{
    // get the header view from section 1
    UITableViewHeaderFooterView *headerview = [self.tableView headerViewForSection: 1];
    [[headerview textLabel] setTextColor: [UIColor redColor]];

    // loop through the subviews
    for (UIView *subview in [headerview subviews]){
        if ([subview isKindOfClass: [UIActivityIndicatorView class]]){
            [subview removeFromSuperview];
        }
    }
}

/*  when the operation successfully finished with the given api response this method gets called    */
- (void) operation:(SBOperation *) operation didFinishWithResult:(SBAPIResponse *) response
{
    // set the tint color of the title header view to red
    UITableViewHeaderFooterView *headerView = [self.tableView headerViewForSection: 1];
    NSDictionary *addressInfo = (NSDictionary *)[response resultDictionary];

    // loop through the subviews to disable the activityindicator
    for (UIView *subview in [headerView subviews]){
        if ([subview isKindOfClass: [UIActivityIndicatorView class]]){
            [subview removeFromSuperview];
        }
    }

    // when the result is success the address is valid
    if ([response objectResultType] == SBAPIResultSuccessType) {

        // set the street and town textfield
        [self.streetField setText: [addressInfo valueForKey: SBZipCodeLookUpKeyStreet]];
        [self.townField setText: [addressInfo valueForKey: SBZipCodeLookUpKeyCity]];

        // set the users address information
        [self.currentUser setZipcode: [addressInfo valueForKey: SBZipCodeLookUpKeyZipCode]];
        [self.currentUser setHouseNumber: [addressInfo valueForKey: SBZipCodeLookUpKeyHouseNumber]];
        [self.currentUser setAddition: [addressInfo valueForKey: SBZipCodeLookUpKeyHouseNumberAddition]];
        [self.currentUser setStreet: [addressInfo valueForKey: SBZipCodeLookUpKeyStreet]];
        [self.currentUser setTown: [addressInfo valueForKey: SBZipCodeLookUpKeyCity]];

        // set the text color
        [[headerView textLabel] setTextColor: [UIColor blackColor]];


    } else {

        // unset address and set the text color of the header to red to show invalid address
        [[headerView textLabel] setTextColor: [UIColor redColor]];
        [self.currentUser unsetAddress];

        // reset the town and address field
        [self.streetField setText: @""];
        [self.townField setText: @""];
    }
}

#pragma mark    -
#pragma mark    UIKeyboard appearance methods

/*  When the keyboard appeared append the frame of the tableview and scroll to the corresponding textfield  */
- (void) keyboardDidAppear:(NSNotification *)notification
{
    // get the frame rectangle and append the frame of the tableview
    CGRect keyboardFrame = [[[notification userInfo] valueForKey: UIKeyboardFrameBeginUserInfoKey] CGRectValue];

    // create null edge insets to scroll the tableview to the top
    UIEdgeInsets contentInsets = UIEdgeInsetsMake(0, 0, keyboardFrame.size.height, 0);
    self.tableView.contentInset = contentInsets;
    self.tableView.scrollIndicatorInsets = contentInsets;

    // get the current view frame and append the height with the eyboard height
```

```
    CGRect currentFrame = self.tableView.frame;
    currentFrame.size.height -= keyboardFrame.size.height;

    // check whether the current visible rectangle shows the textfield
    if (!CGRectContainsPoint(currentFrame, self.currentTextField.superview.frame.origin)) {

        // calculate the point to scroll too within the tableview
        CGPoint scrollPoint = CGPointMake(0, (self.currentTextField.superview.frame.origin.y - currentFrame.size.
            height) + self.currentTextField.frame.size.height);
        [self.tableView setContentOffset: scrollPoint animated:YES];
    }
}

/*  Append the content insets of the tableview when the keyboard will be hidden */
- (void) keyboardWillDisappear:(NSNotification *)notification
{
    // set the tableview scroll contentn to zero and get the animation duration
    UIEdgeInsets contentInsets = UIEdgeInsetsZero;
    CGFloat animationDuration = [[[notification userInfo] objectForKey: UIKeyboardAnimationDurationUserInfoKey]
        floatValue];

    // get the animation curve of the keyboard hiding
    UIViewAnimationCurve animationCurve = [[[notification userInfo] objectForKey:
        UIKeyboardAnimationCurveUserInfoKey] intValue];

    // animate the insets of the tableview
    [UIView animateWithDuration: animationDuration delay: 0 options: (animationCurve << 16) animations:^
    {
        // set the insets to zero with animation
        self.tableView.contentInset = contentInsets;
    } completion: ^(BOOL finished)
    {
        // the scroll indicator insets should not be animated
        self.tableView.scrollIndicatorInsets = contentInsets;
    }];
}

#pragma mark    -
#pragma mark    UITextField Delegate

/*  Gets notified when a specific textfield gets first responder    */
- (void) textFieldDidBeginEditing:(UITextField *)textField
{
    // set the current textfield and mark it as valid
    self.currentTextField = (SBTextField *)textField;

    // when the cell was marked as invalid redraw the cell to make sure the error border is gone
    if (self.currentTextField.isMarkedAsInvalid) {

        // reload the cell as unmarked
        [self.currentTextField setIsMarkedAsInvalid: NO];
        UITableViewCell *correspondingCell = [self.tableView cellForRowAtIndexPath: self.currentTextField.
            correspondingIndexPath];

        // reset the border coloring and width
        correspondingCell.layer.borderWidth = 0.0f;
    }

    // create the layout object
    SBKeyboardToolbarLayout *layout = [SBKeyboardToolbarLayout layoutWithLeftPreviousButtonRightNextButton];
    [layout setTitle: [self.currentTextField toolbarInfo]];

    // when the first textfield is selected do not show the previous button
    if ([textField isEqual: self.firstnameField]) {

        // remove the left bar button item
        [layout setLeftBarButtonItem: nil];

        // create layout object for the firstname field
    } else if ([textField isEqual: self.houseNumberField]) {

        UIBarButtonItem *rightBarButton = [[UIBarButtonItem alloc] initWithTitle:
            NSLocalizedString(@"button.title.continue", nil) style: UIBarButtonItemStylePlain target: nil action:
            nil];
        UIBarButtonItem *leftBarButton = [[UIBarButtonItem alloc] initWithTitle:
            NSLocalizedString(@"button.title.previous", nil) style: UIBarButtonItemStylePlain target: nil action:
            nil];

        // set the new layout
        layout = [SBKeyboardToolbarLayout layoutWithLeftBarButtonItem: leftBarButton rightBarButtonItem:
            rightBarButton title: [self.currentTextField toolbarInfo]];
    }

    // set the new layout type for the toolbar
    [(SBKeyboardToolbar *)[textField inputAccessoryView] redrawWithLayout: layout];
}
```

```
/*  Gets called when the text in a textfield changes, only listen for house number and zipcode  for validation  */
- (BOOL) textFieldDidChangeNotification:(NSNotification *) notification
{
    // only validate the zipcode and house number textfields
    if ([[notification object] isEqual: self.zipcodeField] || [[notification object] isEqual: self.houseNumberField]
        )
    {
        // strip whitespaces out of the zipcode
        if ([[notification object] isEqual: self.zipcodeField]) {
            [self.zipcodeField setText: [[self.zipcodeField text] stringByTrimmingCharactersInSet: [NSCharacterSet
                whitespaceAndNewlineCharacterSet]]];
        }

        // create dictionary for results
        NSDictionary *resultDictionary = nil;

        // cancel currently running operations when already executing
        if (self.addressOperationUUID != nil) {
            [[SBOperationManager defaultManager] cancelOperations: [NSArray arrayWithObject: self.
                addressOperationUUID]];
        }

        // get the header view from section 1
        UITableViewHeaderFooterView *headerview = [self.tableView headerViewForSection: 1];

        // loop through the subviews to find a uiactivity indicator and cancel it, will be added by the new
        for (UIView *subview in [headerview subviews]) {
            if ([subview isKindOfClass: [UIActivityIndicatorView class]]){
                [subview removeFromSuperview];
            }
        }

        // validate the zipcode and house number
        if ([SBUser isValidZipCode: [self.zipcodeField text] withHouseNumber: [self.houseNumberField text] results:
            &resultDictionary])
        {
            // append the textfields with the validated data, Capital zipcode etc
            [self.zipcodeField setText: [resultDictionary valueForKey: SBUserValidationResultKeyZipCode]];
            [self.houseNumberField setText: [resultDictionary valueForKey: SBUserValidationResultKeyMergedNumber]];

            // create the API request for zipcode checking, has a very short timeout
            SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPIAddressLookupRequestType
                timeout: 15];
            [APIRequest setPOSTValue: [resultDictionary objectForKey: SBUserValidationResultKeyZipCode] forKey:
                @"zipcode"];
            [APIRequest setPOSTValue: [resultDictionary objectForKey: SBUserValidationResultKeyHouseNumber] forKey:
                @"house_number"];
            [APIRequest setPOSTValue: [resultDictionary objectForKey: SBUserValidationResultKeyAddition] forKey:
                @"addition"];

            // create options for the request
            NSDictionary *operationInfo = [NSDictionary dictionaryWithObject: SBAPIAddressLookupOperationName
                forKey: SBOperationNameOptionKey];

            // create the operation that will be executed
            SBURLOperation *addressOperation = [[SBURLOperation alloc] initWithAPIRequest: APIRequest options:
                operationInfo];
            [addressOperation setDelegate: self];

            // start the operation
            self.addressOperationUUID = [[SBOperationManager defaultManager] addOperation: addressOperation];

            // append the header to show the spinner execution
            UITableViewHeaderFooterView *headerview = [self.tableView headerViewForSection: 1];
            CGRect textRect = [[headerview textLabel] textRectForBounds: CGRectMake(0, 0, 320, 44)
                limitedToNumberOfLines: 1];

            // create activity indicator and set the corresponding frame of the indicator
            UIActivityIndicatorView *activityIndicator = [[UIActivityIndicatorView alloc]
                initWithActivityIndicatorStyle: UIActivityIndicatorViewStyleGray];
            [activityIndicator setFrame: CGRectMake(textRect.size.width + 10, 0, 44, 44)];
            [activityIndicator startAnimating];
            [headerview addSubview: activityIndicator];
        }
        else {
            // clear the town and street fields, they are not valid because the zipcode / housenumber combination is
                not valid
            [self.streetField setText: @""];
            [self.townField setText: @""];
        }
    }

    // always allow editing of the text
    return YES;
}
#pragma mark    -
#pragma mark    Keyboard toolbar delegate methods
```

```
/*  Go to the previous textfield when called    */
- (void) keyboardToolbarLeftBarButtonTapped:(SBKeyboardToolbar *)keyboardToolbar
{
    // when the current textfield being dismissed is the zipcode field show the customer field next
    if ([self.currentTextField isEqual: self.zipcodeField]) {

        // set the customer cannot be marked as invalid and reload the cell
        self.isCustomerControllerLoadedViaKeyboardNext = YES;
        self.canMarkCustomerSelectionFieldAsInvalid = NO;
        [self.tableView reloadRowsAtIndexPaths: [NSArray arrayWithObject: [NSIndexPath indexPathForRow: 4 inSection:
            0]] withRowAnimation: UITableViewRowAnimationNone];

        // show the customer selection view controller
        [self performSegueWithIdentifier: @"userWillSelectCustomer:" sender: self];
    }
    // when the next textfield can become available
    if ([self.currentTextField.previousTextField canBecomeFirstResponder]) {
        [self.currentTextField.previousTextField becomeFirstResponder];

        // scroll to the next appropriate cell
        [self.tableView scrollToRowAtIndexPath: self.currentTextField.nextTextField.correspondingIndexPath
            atScrollPosition:self.currentTextField.nextTextField.textfieldScrollPosition animated: YES];
        return;
    }

    // when the current textfield can resign first responder continue
    if ([self.currentTextField isFirstResponder] && [self.currentTextField canResignFirstResponder]) {
        [self.currentTextField resignFirstResponder];
    }
}

/*  Go to the next textfield when called    */
- (void) keyboardToolbarRightBarButtonTapped:(SBKeyboardToolbar *)keyboardToolbar
{
    // when the current textfield being dismissed is the telephone 2 field show the customer field next
    if ([self.currentTextField isEqual: self.telephone2Field]) {

        // set the customer cannot be marked as invalid and reload the cell
        self.isCustomerControllerLoadedViaKeyboardNext = YES;
        self.canMarkCustomerSelectionFieldAsInvalid = NO;
        [self.tableView reloadRowsAtIndexPaths: [NSArray arrayWithObject: [NSIndexPath indexPathForRow: 4 inSection:
            0]] withRowAnimation: UITableViewRowAnimationNone];

        // show the customer selection button
        [self performSegueWithIdentifier: @"userWillSelectCustomer:" sender: self];
    }

    // when the next textfield can become the first responder contineu
    if ([self.currentTextField.nextTextField canBecomeFirstResponder]) {

        // scroll to the next cell for the form and make that cell first responder
        [self.tableView scrollToRowAtIndexPath: self.currentTextField.nextTextField.correspondingIndexPath
            atScrollPosition:self.currentTextField.nextTextField.textfieldScrollPosition animated: YES];
        [self.currentTextField.nextTextField becomeFirstResponder];

        return;
    }

    // when the current text field is and can resign first responder
    if ([self.currentTextField isFirstResponder] && [self.currentTextField canResignFirstResponder]) {
        [self.currentTextField resignFirstResponder];

        // when no keyboard is next to be visible the last textfield was edited, validate and go to the next page
        if (![self inputFieldsHaveValidContent]) {

            // continue to the next view controller
            [self performSegueWithIdentifier: @"userWantsToAddMaintenance:" sender: self];
            [self.currentUser save];
        } else {

            // create the alert view to show that not all input values are correct
            UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
                NSLocalizedString(@"login.controller.validation.error.title", nil) message:
                NSLocalizedString(@"login.controller.validation.error.description", nil) delegate: nil
                cancelButtonTitle: nil otherButtonTitles: NSLocalizedString(@"alertview.button.title.dismiss", nil),
                nil];

            // show the alert
            [alertView show];
        }
    }
}

#pragma mark    -
#pragma mark    SBCustomer selector delegate methods
```

```
/*  When the user finish picking the customer this method gets called   */
- (void) customerControllerDidFinishPickingCustomer:(SBPhoneCustomerViewController *)controller
{
    // dismiss the modal view controller
    [self dismissViewControllerAnimated: YES completion: nil];

    // when the texfield made sure the customer controller was shown show the zipcode field
    if (self.isCustomerControllerLoadedViaKeyboardNext && [self.zipcodeField canBecomeFirstResponder]) {
        [self.zipcodeField becomeFirstResponder];
    }

    // reset boolean
    self.isCustomerControllerLoadedViaKeyboardNext = NO;

    // deselect the cell for the customer
    [self.tableView deselectRowAtIndexPath: [self.tableView indexPathForSelectedRow] animated: YES];
}

/*  When the user has selected a new customer this method gets called   */
- (void) customerController:(SBPhoneCustomerViewController *)controller didSelectItem:(SBCustomerDescriptor *)
    descriptor
{
    // set the new customer and hide the modal view controller
    [self dismissViewControllerAnimated: YES completion: nil];

    // when the texfield made sure the customer controller was shown show the zipcode field
    if (self.isCustomerControllerLoadedViaKeyboardNext && [self.zipcodeField canBecomeFirstResponder]) {
        [self.zipcodeField becomeFirstResponder];
    }

    // reset boolean
    self.isCustomerControllerLoadedViaKeyboardNext = NO;

    // set the users customer and reload the cell for the indexpath corresponding to the customer cell
    [self.currentUser setUserCustomer: descriptor];
    [self.tableView reloadRowsAtIndexPaths: [NSArray arrayWithObject: [NSIndexPath indexPathForRow: 4 inSection: 0]]
        withRowAnimation: UITableViewRowAnimationAutomatic];

    // deselect the cell for the customer
    [self.tableView deselectRowAtIndexPath: [self.tableView indexPathForSelectedRow] animated: YES];
}

@end
```

```
//
//  SBPhoneOverviewViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 22-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBMaintenanceInterface.h"
#import "SBURLOperationDelegate.h"
#import "SBOperationDelegate.h"

@class SBReservationManager;

@interface SBPhoneOverviewViewController : UIViewController <SBMaintenanceInterface, UITableViewDataSource,
    UITableViewDelegate, NSFetchedResultsControllerDelegate, SBURLOperationDelegate, SBOperationDelegate>

// the tableview that holds the already or pending request
@property (nonatomic, strong) UITableView *overviewTable;
@property (nonatomic, strong) NSFetchedResultsController *fetchedReservationController;

// boolean indicating whether the states are already being synchronized
@property (nonatomic) BOOL isSynchronizingStates;

// the reservation manager with information about all the reservation objects
@property (nonatomic, strong) SBReservationManager *reservationManager;

@end
```

```
//
//  SBPhoneOverviewViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 22-08-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBPhoneOverviewViewController.h"
#import "SBReservationOverviewCell.h"
#import "SBReservationManager.h"
#import "SBAppDelegate.h"
#import "Reservation.h"
#import "Proceeding.h"
#import "SBVerificationManager.h"
#import "Defect.h"
#import "SBManagedObjectInterface.h"

// new operation classes
#import "SBAPIRequest.h"
#import "SBAPIResponse.h"
#import "SBOperationManager.h"
#import "SBURLSynchronizeOperation.h"
#import "SBURLReservationOperation.h"
#import "SBURLOperation.h"
#import "SBURLCachedImageOperation.h"

// define the string identifiers for different operation types
static NSString *SBAPICancelReservationOperationName            =   @"SBAPICancelReservationOperationName";
static NSString *SBAPISynchronizeOperationName                  =   @"SBAPISynchronizeOperationName";
static NSString *SBAPIUpdateStatesOperationName                 =   @"SBAPIUpdateStatesOperationName";
static NSString *SBAPIUploadReservationOperationName            =   @"SBAPIUploadReservationOperationName";
static NSString *SBAPIFetchImageOperationName                   =   @"SBAPIFetchImageOperationName";

@interface SBPhoneOverviewViewController ()

// method listens for when the new reservation button has been pressed
- (IBAction) shouldCreateNewReservation:(UIBarButtonItem *) sender;

// method creates and executes the url request to synchronize all reservations
- (void) startReservationStateSynchronization;

// gets called when the authentication credentials became available
- (void) authenticationCredentialsDidBecomeAvailable:(NSNotification *) notification;

// process refreshed reservation objects with the new state
- (void) processRefreshedReservationStatesWithDictionary:(NSDictionary *) info;

@end

@implementation SBPhoneOverviewViewController

/*  Override init method to create default values   */
- (id) initWithCoder:(NSCoder *)aDecoder
{
    self = [super initWithCoder: aDecoder];
    if (self) {

        // Custom initialization
        self.reservationManager = [SBReservationManager defaultReservationManager];
    }
    return self;
}

/*  When the view loads do custom initialization for the view content   */
- (void) viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    // create the barbutton item for creating a new reservation
    UIBarButtonItem *newButton = [[UIBarButtonItem alloc] initWithTitle:
        NSLocalizedString(@"overview.controller.navigation.new.title", nil) style: UIBarButtonItemStylePlain target:
        self action: @selector(shouldCreateNewReservation:)];

    // set the bar button item and title of the controller
    [self.navigationItem setRightBarButtonItem: newButton];
    [[self.navigationItem rightBarButtonItem] setEnabled: NO];
    [self setTitle: NSLocalizedString(@"overview.controller.title", nil)];

    // register for the notification that new credentials have been loaded
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector
        (authenticationCredentialsDidBecomeAvailable:) name:
        SBVerificationManagerDidRefreshAuthenticationTokenNotification object: nil];

    // get notified when the application enters the foreground, this way we can start refreshing the reservations
        when needed for the ones in waiting state
```

```
    [[NSNotificationCenter defaultCenter] addObserver: self selector: @selector(viewWillAppear:) name:
        UIApplicationDidBecomeActiveNotification object: nil];

    // validate whether the application can communicate with the server using credentials
    if ([(SBAppDelegate *)[[UIApplication sharedApplication] delegate]
        canCommunicateViaSecureSocketLayersAndAuthentication]) {

        // reenable the right bar button item
        [[self.navigationItem rightBarButtonItem] setEnabled: YES];

        // when the application data has to be synced start that process
        if ([SBURLSynchronizeOperation needsSynchronization]) {

            // create info dictionary for the operation
            NSDictionary *operationInfo = [NSDictionary dictionaryWithObject: SBAPISynchronizeOperationName forKey:
                SBOperationNameOptionKey];

            // create the sync API request
            SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPISynchronizeDefectsRequestType
                timeout: 120];

            // create and configure the sync operation
            SBURLSynchronizeOperation *synchronizeOperation = [[SBURLSynchronizeOperation alloc] initWithAPIRequest:
                APIRequest options: operationInfo];
            [synchronizeOperation setDelegate: self];

            // add to the queue
            [[SBOperationManager defaultManager] addOperation: synchronizeOperation];
        }
    }
}

/*  Respond to low memory contitions    */
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];

    // when not on screen try to release some used memory
    if ([self.view window] == nil) {

        // when the tableview is on screen remove it from superview
        if ([self.overviewTable superview] != nil) {
            [self.overviewTable removeFromSuperview];
        }

        // release the fetched results controller and tableview
        self.overviewTable = nil;
        self.fetchedReservationController = nil;
    }
}

/*  When proceeding to the next view controller make sure to set the delegate object    */
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([[segue identifier] isEqualToString: @"createNewReservation:"]) {

        // let the manager create a new reservation object
        //[self.reservationManager wizardWillGenerateReservation: [segue destinationViewController]];
        [self.reservationManager setCurrentProceedingIndex: 0];

        // set the delegate of the navigation controller
        [[segue destinationViewController] setDelegate: self];
    }
}

/*  Sart synchronization of states when the view will become visible    */
- (void) viewWillAppear:(BOOL)animated
{
    // when the table view is not yet created
    if (self.overviewTable == nil) {

        // create the tableview for holding the different reservation objects
        self.overviewTable = [[UITableView alloc] initWithFrame: CGRectZero style: UITableViewStylePlain];
        [self.overviewTable registerNib: [UINib nibWithNibName: @"SBReservationOverviewCell" bundle: nil]
            forCellReuseIdentifier: @"reservationOverviewCell"];

        // set delegate and datasource
        [self.overviewTable setDataSource: self];
        [self.overviewTable setDelegate: self];
    }

    // when the fetched results controller has not been created do it here
    if (self.fetchedReservationController == nil) {

        // create the reservation controller object
        NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] initWithEntityName: @"Reservation"];
        NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey: @"currentStatus" ascending: YES];
```

```
        [fetchRequest setSortDescriptors: [NSArray arrayWithObject: sortDescriptor]];
        NSManagedObjectContext *context = [(SBAppDelegate *)[[UIApplication sharedApplication] delegate]
            managedObjectContext];

        // do not use cache or section key path
        self.fetchedReservationController = [[NSFetchedResultsController alloc] initWithFetchRequest: fetchRequest
            managedObjectContext:context sectionNameKeyPath: nil cacheName: nil];
        [self.fetchedReservationController setDelegate: self];

        // create error and start fetching the
        NSError *error = nil;
        if (![self.fetchedReservationController performFetch: &error]) {
            NSLog(@"Failed fetching reservations: %@", error);
        }
    }

    // do an initial state check
    if (!self.isSynchronizingStates) {
        [self startReservationStateSynchronization];
    }

    // when the tableview is not yet added to a superview add it to this view
    if (self.overviewTable.superview == nil) {

        // set the bounds of the tableview and add to view
        [self.overviewTable setFrame: self.view.bounds];
        [self.view addSubview: self.overviewTable];
    }
}

#pragma mark    -
#pragma mark    IBaction methods

/*  MEthod gets called when the user presses the button that will show the reservation create wizard    */
- (IBAction) shouldCreateNewReservation:(UIBarButtonItem *)sender
{
    // perform the segue to the next view controller
    [self performSegueWithIdentifier: @"createNewReservation:" sender: self];
}

/*  when the credentials become available reenable the right bar button */
- (void) authenticationCredentialsDidBecomeAvailable:(NSNotification *)notification
{
    // when not already synchronizing return
    if (![SBURLSynchronizeOperation needsSynchronization]) {
        return;
    }

    // enable the right bar button item
    [[self.navigationItem rightBarButtonItem] setEnabled: YES];

    // create info dictionary for the operation
    NSDictionary *operationInfo = [NSDictionary dictionaryWithObject: SBAPISynchronizeOperationName forKey:
        SBOperationNameOptionKey];

    // create the sync API request
    SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPISynchronizeDefectsRequestType
        timeout: 120];

    // create and configure the sync operation
    SBURLSynchronizeOperation *synchronizeOperation = [[SBURLSynchronizeOperation alloc] initWithAPIRequest:
        APIRequest options: operationInfo];
    [synchronizeOperation setDelegate: self];

    // start the operation execution
    [[SBOperationManager defaultManager] addOperation: synchronizeOperation];
}

/*  start the state synchronization process for all reservations that have a state which waits for approval etc.
    */
- (void) startReservationStateSynchronization
{
    // retrieve all current reservation objects
    NSArray *allReservations = [[[self.fetchedReservationController sections] objectAtIndex: 0] objects];
    NSEnumerator *reservationEnumerator = [allReservations objectEnumerator];
    Reservation *currentReservation = nil;

    // create the mutable string that holds all external IDs
    NSMutableString *IDString = [[NSMutableString alloc] init];
    BOOL shouldStartSynchronization = NO;

    // add all reservation external IDs to the POST string that are in the following states:
    //  -   Approved
    //  -   Waiting for approval
    while (currentReservation = [reservationEnumerator nextObject]) {

        if ([currentReservation externalReservationId] != nil && ([[currentReservation currentStatus] integerValue]
```

```
            == SBReservationStatusApproved || [[currentReservation currentStatus] integerValue] ==
            SBReservationStatusWaiting)) {

            // add a comma seperator when a previous ID has been inserted
            if ([IDString length] > 0) {
                [IDString appendString: @","];
            }

            // add the external reservation id to the array
            [IDString appendString: [NSString stringWithFormat: @"%@", currentReservation.externalReservationId]];
            shouldStartSynchronization = YES;
        }
    }

    // do not start the synchronization when no reservation has to be updated
    if (!shouldStartSynchronization) {
        return;
    }

    // create info dictionary for the operation
    NSDictionary *operationInfo = [NSDictionary dictionaryWithObject: SBAPIUpdateStatesOperationName forKey:
        SBOperationNameOptionKey];

    // create the API request for state checking and add the POST value with ID's that needs syncing to this request
    SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPICheckStateRequestType timeout: 60];
    [APIRequest setPOSTValue: IDString forKey: @"reservation_list"];

    // create the connection operation and configure as delegate
    SBURLOperation *stateOperation = [[SBURLOperation alloc] initWithAPIRequest: APIRequest options: operationInfo];
    [stateOperation setDelegate: self];

    // start the execution of this operation
    [[SBOperationManager defaultManager] addOperation: stateOperation];
}

#pragma mark    -
#pragma mark    UITableView Delegate & datasource methods

/*  This tableview holds one section    */
- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

/*  Return the amount of objects that need to be displayed in the tableview */
- (NSInteger) tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    // return the amount of reservations in the array
    return [[[self.fetchedReservationController sections] objectAtIndex: 0] numberOfObjects];
}

/*  Return the custom tableview cell for this indexpath */
- (UITableViewCell *) tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // create the reuse identifier and get a cell from the dequeue
    static NSString *reuseIdentifier = @"reservationOverviewCell";
    SBReservationOverviewCell *cell = (SBReservationOverviewCell *)[self.overviewTable
        dequeueReusableCellWithIdentifier: reuseIdentifier];

    // when the cell image is not set
    if (cell.imageView.image == nil) {

        // get the reservation managed object
        Reservation *managedObject = [self.fetchedReservationController objectAtIndexPath: indexPath];
        Defect *selectedDefect = [[[managedObject proceedings] anyObject] defect];

        // when a defect is available download the image
        if (selectedDefect != nil && ![selectedDefect imageIsLoaded]) {

            // create info dictionary for the operation
            NSDictionary *operationInfo = [NSDictionary dictionaryWithObject: SBAPIFetchImageOperationName forKey:
                SBOperationNameOptionKey];

            // create operation and configure for loading cached image
            SBURLCachedImageOperation *imageOperation = [[SBURLCachedImageOperation alloc] initWithManagedObjectID:
                [selectedDefect objectID] withOptions: operationInfo];
            [imageOperation setDelegate: self];

            // add the image operation to the operation queue
            [[SBOperationManager defaultManager] addOperation: imageOperation];

        } else if ([selectedDefect imageIsLoaded] && selectedDefect.fetchedImage != nil){

            // set the loaded image from the defect
            [[cell reservationImage] setImage: selectedDefect.fetchedImage];

        } else {
```

```
            // set the default image
            [[cell reservationImage] setImage: [UIImage imageNamed: @"defectNotFound"]];
        }
    }

    // set the reservation object of the cell
    [cell setValue: [self.fetchedReservationController objectAtIndexPath: indexPath] forKey: @"reservation"];

    // no selection style for the cell
    [cell setSelectionStyle: UITableViewCellSelectionStyleNone];

    return cell;
}

/*  Specify the height of all the cells */
- (CGFloat) tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return 100.0;
}

/*  Gets called when the user pressed the edit button in the cell   */
- (void) tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
    forRowAtIndexPath:(NSIndexPath *)indexPath
{
    // when the delete button was pressed check whether this reservation can be deleted
    if (editingStyle == UITableViewCellEditingStyleDelete) {

        // get the reservation object that is selected
        Reservation *selectedReservation = [self.fetchedReservationController objectAtIndexPath: indexPath];
        NSInteger reservationStatus = [[selectedReservation currentStatus] integerValue];

        // for the approved and waiting status the communication with the server must be handled first
        if (reservationStatus == SBReservationStatusApproved || reservationStatus == SBReservationStatusWaiting) {

            // create info dictionary for the operation
            NSDictionary *operationInfo = [NSDictionary dictionaryWithObject: SBAPICancelReservationOperationName
                forKey: SBOperationNameOptionKey];

            // create API request for cancelling a reservation and add the POST cancel value
            SBAPIRequest *APIRequest = [[SBAPIRequest alloc] initWithRequestCall: SBAPICancelReservationRequestType
                timeout: 60];
            [APIRequest setPOSTValue: [selectedReservation externalReservationId] forKey: @"reservation_id"];

            // create and configure the operation
            SBURLOperation *cancelOperation = [[SBURLOperation alloc] initWithAPIRequest: APIRequest options:
                operationInfo];
            [cancelOperation setDelegate: self];

            // add the operation to the execution queue
            [[SBOperationManager defaultManager] addOperation: cancelOperation];
        }
        else {

            // reset the current reservation object in the manager because it will be deleted
            if ([selectedReservation isEqual: [self.reservationManager currentReservation]]) {
                [self.reservationManager setCurrentReservation: nil];
            }
            // the object can be deleted from the managed object context
            NSManagedObjectContext *context = [(SBAppDelegate *)[[UIApplication sharedApplication] delegate]
                managedObjectContext];

            // delete the selected reservation object from the store
            [context deleteObject: selectedReservation];
        }
    }

    // return to the not editing mode of the tableview
    [tableView setEditing: NO animated: YES];
}

/*  Show the delete / cancel title for the editing button   */
- (NSString *) tableView:(UITableView *)tableView titleForDeleteConfirmationButtonForRowAtIndexPath:(NSIndexPath *)
    indexPath
{
    // get the object for which the title will be shown
    Reservation *currentReservation = [self.fetchedReservationController objectAtIndexPath: indexPath];
    NSInteger reservationstatus = [[currentReservation currentStatus] integerValue];

    // when the status is approved or waiting show the cancel title
    if (reservationstatus == SBReservationStatusApproved || reservationstatus == SBReservationStatusWaiting) {
        return NSLocalizedString(@"reservation.tableview.cancel.title", nil);
    }
    else {
        // show the normal delete title
        return NSLocalizedString(@"reservation.tableview.delete.title", nil);
    }
```

```
}

#pragma mark    -
#pragma mark    Delegate of the wizard

/*  When the wizard will show a new proceeding object   */
- (void) willShowProceedingForIndex:(NSInteger) index
{
    // set the current proceeding object being edited
    [self.reservationManager setCurrentProceedingIndex: index];
}

/*  When the controller wizard changes a value of the proceeding this method commits it to the manager of the
    reservation being created or edited   */
- (void) viewController:(UIViewController *)viewController didChangeValue:(id)newValue forProceedingKey:(NSString *)
    key
{
    // let the reservation manager process the value and key combination
    [self.reservationManager processValue: newValue forCurrentProceedingPropertyKey: key];
}

/*  When the wizard completed creating the reservation this method will dismiss the wizard and start the
    validation / upload process    */
- (void) viewControllerDidFinishCreatingReservation:(UIViewController *) viewController
{
    // dismiss the wizard modal view controller
    [self dismissViewControllerAnimated: YES completion:^
    {
        // create info dictionary for the operation
        NSDictionary *operationInfo = [NSDictionary dictionaryWithObject: SBAPIUploadReservationOperationName
            forKey: SBOperationNameOptionKey];

        // create the reservation upload operation nd configure it
        SBURLReservationOperation *reservationUploader = [[SBURLReservationOperation alloc] initWithReservation:
            [self.reservationManager currentReservation] options: operationInfo];
        [reservationUploader setDelegate: self];

        // set uploading status
        [[self.reservationManager currentReservation] setCurrentStatus: [NSNumber numberWithInt:
            SBReservationStatusUploading]];

        // start the upload operation
        [[SBOperationManager defaultManager] addOperation: reservationUploader];
        //[self.reservationManager wizardDidCompleteGeneratingReservation: viewController];
    }];
}

/*  Returns an existing value for the current proceeding being shown    */
- (id) viewController:(UIViewController *)viewController proceedingValueForKey:(NSString *)key
{
    // when the user is requested return the user object for the current reservation being edited
    if ([key isEqualToString: @"reservationUser"]) {
        return [self.reservationManager userForCurrentReservation];
    }

    // get the value and return it
    return [self.reservationManager valueForCurrentProceedingProperty: key];
}

/*  Returns the reservation object for the overview form    */
- (Reservation *) viewControllerWillShowReservationOverview:(UIViewController *) viewController
{
    // this method gets called when the wizard will eventually show the reservation overview of the current created
        reservation object
    return [self.reservationManager currentReservation];
}

/*  When the wizrad will create a new proceeding increase the array counter, automatically creates a new proceeding
    */
- (void) viewControllerWillCreateNewProceeding:(UIViewController *)controller
{
    // increase the amount of proceedings currently in the array by 1
    [self.reservationManager setCurrentProceedingIndex: ([self.reservationManager currentProceedingIndex] + 1)];
}

#pragma mark    -
#pragma mark    NSFetchedResultsController delegate

/*  When the controller will change its contentn this method gets called en enables the tableview edit  */
- (void) controllerWillChangeContent:(NSFetchedResultsController *)controller
{
    // start updating the tableview
    [self.overviewTable beginUpdates];
}

/*  When the controller changes finished end the tableview editing  */
- (void) controllerDidChangeContent:(NSFetchedResultsController *)controller
```

```
{
    // end updating the tableview
    [self.overviewTable endUpdates];
}

/*  The specific insert, delete, update or move actions are executed in this method */
- (void) controller:(NSFetchedResultsController *)controller didChangeObject:(id)anObject atIndexPath:(NSIndexPath *
    )indexPath forChangeType:(NSFetchedResultsChangeType)type newIndexPath:(NSIndexPath *)newIndexPath
{
    // switch to the correct updating type
    switch (type) {
        // when an new reservation is inserted animate it from the top of the tableview
        case NSFetchedResultsChangeInsert:
            [self.overviewTable insertRowsAtIndexPaths: [NSArray arrayWithObject: newIndexPath] withRowAnimation:
                UITableViewRowAnimationTop];
            break;

        // when a reservation is deleted fade it out of the tableview
        case NSFetchedResultsChangeDelete:
            [self.overviewTable deleteRowsAtIndexPaths: [NSArray arrayWithObject: indexPath] withRowAnimation:
                UITableViewRowAnimationTop];
            break;

        // when a reservation is updated just update the row with no animation
        case NSFetchedResultsChangeUpdate:
            [self.overviewTable reloadRowsAtIndexPaths: [NSArray arrayWithObject: indexPath] withRowAnimation:
                UITableViewRowAnimationNone];
            break;

        // when rows are moved commit that move to the tableview
        case NSFetchedResultsChangeMove:
            [self.overviewTable moveRowAtIndexPath: indexPath toIndexPath: newIndexPath];
            break;
    }
}

/*  When a section of the fetched results controller changes this method gets called    */
- (void) controller:(NSFetchedResultsController *)controller didChangeSection:(id<NSFetchedResultsSectionInfo>)
    sectionInfo atIndex:(NSUInteger)sectionIndex forChangeType:(NSFetchedResultsChangeType)type
{
    // do nothing, no sections in our tableview
}

#pragma mark    -
#pragma mark    SBOperation delegate

/*  Processes the data that was received for all reservations   */
- (void) processRefreshedReservationStatesWithDictionary:(NSDictionary *)info
{
    // retrieve the synchronized objects from the dictionary
    NSEnumerator *arrayEnumerator = [info objectEnumerator];

    // retreive all objects from the fetched results controller
    NSArray *reservations = [[[self.fetchedReservationController sections] objectAtIndex: 0] objects];
    NSDictionary *currentSyncedObject = nil;

    // loop through all objects that need to be synced
    while (currentSyncedObject = [arrayEnumerator nextObject]) {

        NSNumber *currentReservationId = [currentSyncedObject objectForKey: @"id"];
        NSString *currentStateDescription = [currentSyncedObject objectForKey: @"state"];

        // create the enumerator for the reservation objects
        NSEnumerator *reservationEnumerator = [reservations objectEnumerator];
        Reservation *currentReservation = nil;

        // loop through all reservations to find the object with the corresponding external ID
        while (currentReservation = [reservationEnumerator nextObject]) {

            // process the new state when the external reservation id is equal
            if ([[currentReservation externalReservationId] isEqualToNumber: currentReservationId]) {

                if ([currentStateDescription isEqualToString: @"reserved"]) {
                    [currentReservation setCurrentStatus: [NSNumber numberWithInteger: SBReservationStatusWaiting]];
                }
                else if ([currentStateDescription isEqualToString: @"approved"]) {
                    [currentReservation setCurrentStatus: [NSNumber numberWithInteger: SBReservationStatusApproved]]
                        ;
                }
                else if ([currentStateDescription isEqualToString: @"denied"]) {
                    [currentReservation setCurrentStatus: [NSNumber numberWithInteger: SBReservationStatusDenied]];
                }
                else if ([currentStateDescription isEqualToString: @"finished"]) {
                    [currentReservation setCurrentStatus: [NSNumber numberWithInteger: SBReservationStatusFinished]]
                        ;
                }
                else if ([currentStateDescription isEqualToString: @"cancelled"]) {
```

```
                    [currentReservation setCurrentStatus: [NSNumber numberWithInteger: SBReservationStatusCancelled]
                        ];
                }
            }
        }
    }
}

#pragma mark    -
#pragma mark    New operation delegate respond methods

/*  When the operation failed to execute this method is called with the corresponding error */
- (void) operation:(SBOperation *)operation didFailWithError:(NSError *)error
{
    // get the operation name of the failed operation
    NSString *operationName = [[operation options] objectForKey: SBOperationNameOptionKey];

    // check which operation has failed execution
    if ([operationName isEqualToString: SBAPISynchronizeOperationName]) {
        NSLog(@"Failed synchronization with error: %@", error);
    } else if ([operationName isEqualToString: SBAPICancelReservationOperationName]) {

        // create alertview that identifies the cancelling failed
        UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
            NSLocalizedString(@"operation.cancel.reservation.failed.title", nil) message:
            NSLocalizedString(@"operation.cancel.reservation.failed.description", nil) delegate: nil
            cancelButtonTitle: nil otherButtonTitles: NSLocalizedString(@"alertview.button.title.dismiss", nil), nil
            ];

        // show the error, no further actions have to be taken
        [alertView show];

    } else if ([operationName isEqualToString: SBAPIUpdateStatesOperationName]) {
        NSLog(@"Failed reloading states with error: %@", error);
    } else if ([operationName isEqualToString: SBAPIFetchImageOperationName]) {

        // cast the object as a cached image operation to fetch the object ID
        SBURLCachedImageOperation *cacheOperation = (SBURLCachedImageOperation *)operation;
        Reservation<SBManagedObjectInterface> *managedObject = (Reservation<SBManagedObjectInterface> *)
            [[(SBAppDelegate *)[[UIApplication sharedApplication] delegate] managedObjectContext] objectWithID:
            cacheOperation.objectID];

        // get the indexpath of the row that needs to be reloaded
        [managedObject setImageLoaded: YES];

        // get all fetched objects and try to find the corresponding defect
        NSEnumerator *objectEnumerator = [[self.fetchedReservationController fetchedObjects] objectEnumerator];
        Reservation *currentReservation = nil;

        // try to find the corresponding defect
        while (currentReservation = [objectEnumerator nextObject]) {

            // create enumerator for the proceedings of the reservation
            NSEnumerator *proceedingEnumerator = [[currentReservation proceedings] objectEnumerator];
            Proceeding *currentProceeding = nil;

            // find the corresponding defect
            while (currentProceeding = [proceedingEnumerator nextObject]) {

                // when the defect equals the finished object
                if ([[currentProceeding defect] isEqual: managedObject]) {

                    // get the indexpath that needs to be updated
                    NSIndexPath *correspondingIndexPath = [self.fetchedReservationController indexPathForObject:
                        currentReservation];
                    [self.overviewTable reloadRowsAtIndexPaths: [NSArray arrayWithObject: correspondingIndexPath]
                        withRowAnimation: UITableViewRowAnimationNone];
                }
            }
        }


    } else if ([operationName isEqualToString: SBAPIUploadReservationOperationName]) {

        // alert the user that the reservation could not be uploaded
        UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
            NSLocalizedString(@"operation.reservation.upload.failed.title", nil) message:
            NSLocalizedString(@"operation.reservation.upload.failed.description", nil) delegate: nil
            cancelButtonTitle: nil otherButtonTitles: NSLocalizedString(@"alertview.button.title.dismiss", nil), nil
            ];

        // show the view
        [alertView show];
    }
}

/*  When the operation changes to a next state this method gets called  */
```

```
- (void) operation:(SBOperation *) operation didChangeToState:(SBOperationStateInterface *) newState
{

}

/*  When the operation finishes with a response from the API server this method gets called */
- (void) operation:(SBOperation *) operation didFinishWithResult:(SBAPIResponse *) response
{
    // get the operation name of the failed operation
    NSString *operationName = [[operation options] objectForKey: SBOperationNameOptionKey];

    if ([operationName isEqualToString: SBAPIUpdateStatesOperationName]) {

        // when successfully executed the API request process the results
        if ([response objectResultType] == SBAPIResultSuccessType) {
            [self processRefreshedReservationStatesWithDictionary: (NSDictionary *)[response resultDictionary]];
        }
    } else if ([operationName isEqualToString: SBAPIFetchImageOperationName]) {

        // cast the object as a cached image operation to fetch the object ID
        SBURLCachedImageOperation *cacheOperation = (SBURLCachedImageOperation *)operation;
        Reservation<SBManagedObjectInterface> *managedObject = (Reservation<SBManagedObjectInterface> *)
            [[(SBAppDelegate *)[[UIApplication sharedApplication] delegate] managedObjectContext] objectWithID:
            cacheOperation.objectID];

        // get the indexpath of the row that needs to be reloaded
        [managedObject setImageLoaded: YES];

        // get all fetched objects and try to find the corresponding defect
        NSEnumerator *objectEnumerator = [[self.fetchedReservationController fetchedObjects] objectEnumerator];
        Reservation *currentReservation = nil;

        // try to find the corresponding defect
        while (currentReservation = [objectEnumerator nextObject]) {

            // create enumerator for the proceedings of the reservation
            NSEnumerator *proceedingEnumerator = [[currentReservation proceedings] objectEnumerator];
            Proceeding *currentProceeding = nil;

            // find the corresponding defect
            while (currentProceeding = [proceedingEnumerator nextObject]) {

                // when the defect equals the finished object
                if ([[currentProceeding defect] isEqual: managedObject]) {

                    // get the indexpath that needs to be updated
                    NSIndexPath *correspondingIndexPath = [self.fetchedReservationController indexPathForObject:
                        currentReservation];
                    [self.overviewTable reloadRowsAtIndexPaths: [NSArray arrayWithObject: correspondingIndexPath]
                        withRowAnimation: UITableViewRowAnimationNone];
                }
            }
        }
    } else if ([operationName isEqualToString: SBAPICancelReservationOperationName]) {

        // check whether the API request is executed successfully
        if ([response objectResultType] == SBAPIResultSuccessType && [[response resultDictionary] objectForKey:
            @"id"] != nil) {

            // for formatting the reservation id
            NSNumberFormatter *numberFormatter = [[NSNumberFormatter alloc] init];

            // find the reservation with the received object ID
            NSEnumerator *reservationEnumerator = [[self.fetchedReservationController fetchedObjects]
                objectEnumerator];
            Reservation *currentReservation = nil;
            NSNumber *reservationId = [numberFormatter numberFromString: [[response resultDictionary] objectForKey:
                @"id"]];

            // loop until the correct ID is found
            while (currentReservation = [reservationEnumerator nextObject]) {

                // when the
                if ([[currentReservation externalReservationId] isEqualToNumber: reservationId]) {

                    // get the managed object context
                    NSManagedObjectContext *managedObjectContext = [(SBAppDelegate *)[[UIApplication
                        sharedApplication] delegate] managedObjectContext];

                    // delete the object from the context and break out of looping
                    [managedObjectContext deleteObject: currentReservation];
                    break;
                }
            }

        } else {
```

```
            // create alertview that identifies the cancelling failed
            UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:
                NSLocalizedString(@"operation.cancel.reservation.failed.title", nil) message:
                NSLocalizedString(@"operation.cancel.reservation.failed.description", nil) delegate: nil
                cancelButtonTitle: nil otherButtonTitles: NSLocalizedString(@"alertview.button.title.dismiss", nil),
                nil];

            // show the error, no further actions have to be taken
            [alertView show];
        }
    }
}

@end
```

```
//
//  SBPhonePageRoomViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 26-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "SBPhoneElementViewController.h"
#import "SBCollectionController.h"
#import "Space.h"


@interface SBPhoneSpaceViewController : UIViewController <SBCollectionControllerDelegate>

// the selected id of the previous view Controller, an element
@property (nonatomic, strong) NSNumber *locationId;
@property (nonatomic, strong) SBCollectionController *collectionController;

// the current selected space id
@property (nonatomic, strong) Space *currentSpace;

@end
```

```
//
//  SBPhonePageRoomViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 26-06-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBPhoneSpaceViewController.h"
#import "SBMaintenanceInterface.h"

@implementation SBPhoneSpaceViewController

/*  Initialize the view when loaded from memory */
- (void) viewDidLoad
{
    [super viewDidLoad];

    // set the title of the space controller
    [self setTitle: NSLocalizedString(@"space.controller.title", nil)];
}

/*  In this method tell the delegate to change its NavigationBar    */
- (void) viewWillAppear:(BOOL)animated
{
    // call super with the same method
    [super viewWillAppear: animated];

    // create the collection controller when no instantiated
    if (self.collectionController == nil) {

        // create the fetch request for the fetched results controller, the predicate will only select items with a
            parent location id that is passed to this view controller by the location selection controller
        NSPredicate *spacePredicate = [NSPredicate predicateWithFormat: @"location.locationId == %d", [self.
            locationId intValue]];
        NSSortDescriptor *spaceSortDescriptor = [NSSortDescriptor sortDescriptorWithKey: @"spaceId" ascending: YES];

        // create the fetch request and add the sort descriptor and predicate
        NSFetchRequest *spaceRequest = [[NSFetchRequest alloc] initWithEntityName: @"Space"];
        [spaceRequest setPredicate: spacePredicate];
        [spaceRequest setSortDescriptors: [NSArray arrayWithObject: spaceSortDescriptor]];

        // create the collection controller for all spaces
        self.collectionController = [[SBCollectionController alloc] initWithFrame: CGRectZero fetchRequest:
            spaceRequest];
        [self.collectionController setDelegate: self];

        // get the current selected location id
        id<SBMaintenanceInterface> controller = (id<SBMaintenanceInterface>)[self.navigationController delegate];
        self.currentSpace = (Space *)[controller viewController: self proceedingValueForKey: @"space"];
    }

    // add the collection controller to the view
    if ([self.collectionController superview] == nil) {

        [self.collectionController setFrame: self.view.bounds];
        [self.view addSubview:self.collectionController];
    }
}

/*  When a low memory warning is received release objects that can be recreated */
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];

    // when the view is currently on screen remove objects, else do nothing
    if ([self.view window] == nil) {

        // remove the view from its superview when needed
        if ([self.collectionController superview] != nil) {
            [self.collectionController removeFromSuperview];
        }

        // release the controller
        self.collectionController = nil;
    }
}

#pragma mark    -
#pragma mark    Class methods

/*  Prepare for the next viewCOntroller */
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // when the next controller is the elements controller continue
    if ([[segue identifier] isEqualToString: @"userSelectedSpace:"])
    {
```

```
        // cast the next controller and set the space id
        SBPhoneElementViewController *controller = (SBPhoneElementViewController *) [segue destinationViewController
            ];
        [controller setSpaceId: [self.collectionController identifierForSelectedIndexPath]];

        // commit the selected value
        id<SBMaintenanceInterface> delegate = (id<SBMaintenanceInterface>) [self.navigationController delegate];
        [delegate viewController: self didChangeValue: [self.collectionController managedObjectForSelectedIndexPath]
            forProceedingKey: @"space"];
    }
}

#pragma mark    -

/*  When an item gets selected  */
- (void) collectionController:(SBCollectionController *)controller didSelectItemWithIdentifier:(NSNumber *)
    identifier
{
    [self performSegueWithIdentifier: @"userSelectedSpace:" sender: self];
}

/*  When an item gets deselected    */
- (void) collectionControllerDidDeselectCollectionItem:(SBCollectionController *)controller
{

}



@end
```

```
//
//  SBReservationOverviewViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBViewController.h"
#import "SBURLOperationDelegate.h"
#import "SBOperationDelegate.h"

@class Reservation;

@interface SBReservationOverviewViewController : SBViewController <UITableViewDataSource, UITableViewDelegate,
    SBOperationDelegate, SBURLOperationDelegate, UIAlertViewDelegate>

// the reservation object which should be overviewed in this view controller
@property (nonatomic, strong) Reservation *reservation;

// whether the view controller should be in a modus where the user has to confirm it and upload it (YES). Defaults
    to NO
@property (nonatomic) BOOL reservationRequiresConfirmation;

// the tableview with all reservation informative fields
@property (nonatomic, weak) IBOutlet UITableView *informationTableView;

// a method called from the fetched results controller indicating that the object changed values
- (void) refreshReservation;

@end
```

```
//
//  SBReservationOverviewViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 23-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBReservationOverviewViewController.h"
#import "SBColorManager.h"
#import "SBNavigationBar.h"
#import "SBNavigationBarDetailView.h"
#import "Reservation.h"
#import "Proceeding.h"
#import "SBDayPartDescriptor.h"
#import "Defect.h"
#import "Location.h"
#import "Space.h"
#import "Element.h"
#import "SBReservationManager.h"
#import "SBDefectImageDescriptor.h"
#import "SBReservationPopAnimatedTransitioning.h"
#import "SBURLOperation.h"
#import "SBOperationManager.h"
#import "SBAPIRequest.h"
#import "SBAPIResponse.h"
#import "SBViewCompatabilityFactory.h"
#import "SBSelectableOptionTableViewCell.h"

@interface SBReservationOverviewViewController ()

// when the cancel button is pressed this method is called
- (void) cancelButtonPressed:(UIBarButtonItem *) sender;

// when the confirm button is pressed the reservation should be completed and uploaded
- (void) confirmButtonPressed:(UIBarButtonItem *) sender;

// when the user wants to retry uploading
- (void) retryUploadPressed:(UIBarButtonItem *) sender;

// returns the size for the detail cell
- (CGFloat) heightForDetails;

// gets aclled when the user wants to cancel the reservation, only happens when the reservation is uploaded
- (void) cancelUploadedReservation:(UIBarButtonItem *) sender;

// the text label and image view used for the detail cell
@property (nonatomic, strong) UILabel *detailLabel;
@property (nonatomic, strong) UIImageView *detailImage;

@end

// define tags for alertview types
#define kSBOverviewControllerAlertViewTagCancelNew      1
#define kSBOVerviewControllerAlertViewTagCancelExisting 2

@implementation SBReservationOverviewViewController

/*  Initializer method from the uistoryboard instance   */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    // call super initializer
    self = [super initWithCoder: aDecoder];

    if (self) {

        // set default reservation confirmation value
        self.reservationRequiresConfirmation = NO;
    }

    return self;
}

/*  When the view is loaded in memory do any interface initialization in this method    */
- (void) viewDidLoad
{
    // call super implementation
    [super viewDidLoad];

    // set the title of the view controller
    if (self.reservationRequiresConfirmation) {
        [self setTitle: NSLocalizedString(@"overview.viewcontroller.final.title", nil)];
    } else {
        [self setTitle: NSLocalizedString(@"overview.viewcontroller.title", nil)];
    }

    [self.informationTableView registerNib: [UINib nibWithNibName: @"SBSelectableOptionTableViewCell" bundle: nil]
```

```
        forCellReuseIdentifier: @"FirstFourCells"];

    // when the reservation should be confirmed a different toolbar is shown
    if (self.reservationRequiresConfirmation) {

        // create the cancel button and the horizantal spacing between buttons
        UIBarButtonItem *cancelBarButton = [[UIBarButtonItem alloc] initWithTitle:
            NSLocalizedString(@"overview.viewcontroller.toolbar.cancel.title", nil) style: UIBarButtonItemStylePlain
            target: self action: @selector(cancelButtonPressed:)];
        [cancelBarButton setTintColor: [UIColor lightGrayColor]];
        [cancelBarButton setTitleTextAttributes: [NSDictionary dictionaryWithObject: [UIFont systemFontOfSize: 13.0f
            ] forKey: UITextAttributeFont] forState: UIControlStateNormal];

        // create the save button
        UIButton *confirmButton = [[UIButton alloc] initWithFrame: CGRectMake(160, 0, 160, 44)];
        [confirmButton addTarget: self action: @selector(confirmButtonPressed:) forControlEvents:
            UIControlEventTouchUpInside];
        [confirmButton setBackgroundColor: [[SBColorManager defaultColors] requestNavigationColor]];
        [[confirmButton titleLabel] setFont: [UIFont systemFontOfSize: 13.0f]];
        [confirmButton setTitle: NSLocalizedString(@"overview.viewcontroller.toolbar.confirmation.title", nil)
            forState: UIControlStateNormal];

        // create flexible and fixed space between bar button items
        UIBarButtonItem *flexibleSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
            UIBarButtonSystemItemFlexibleSpace target: nil action: nil];
        UIBarButtonItem *fixedSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
            UIBarButtonSystemItemFixedSpace target: nil action: nil];
        fixedSpace.width -= 16.0f;

        // create the confirm bar button
        UIBarButtonItem *confirmBarbutton = [[UIBarButtonItem alloc] initWithCustomView: confirmButton];

        // set the new toolbar items
        [self.navigationController setToolbarHidden: NO animated: YES];
        [self setToolbarItems: [NSArray arrayWithObjects: cancelBarButton, flexibleSpace, confirmBarbutton,
            fixedSpace, nil] animated: YES];
    } else if ([[self.reservation isUploaded] boolValue] && ([[self.reservation currentStatus] integerValue] ==
        SBReservationStatusApproved || [[self.reservation currentStatus] integerValue] == SBReservationStatusWaiting
        )) {

        // create a bar button item for the cancelling of the reservation when the reservation is uploaded
        UIButton *cancelReservationButton = [[UIButton alloc] initWithFrame: CGRectMake(160, 0, 160, 44)];
        [cancelReservationButton addTarget: self action: @selector(cancelUploadedReservation:) forControlEvents:
            UIControlEventTouchUpInside];
        [cancelReservationButton setBackgroundColor: [[SBColorManager defaultColors] redColor]];
        [[cancelReservationButton titleLabel] setFont: [UIFont systemFontOfSize: 13.0f]];
        [cancelReservationButton setTitle:
            NSLocalizedString(@"overview.viewcontroller.toolbar.cancel.uploaded.title", nil) forState:
            UIControlStateNormal];

        // create the bar button from the other cancel button
        UIBarButtonItem *cancelBarbuttonItem = [[UIBarButtonItem alloc] initWithCustomView: cancelReservationButton]
            ;
        UIBarButtonItem *flexibleSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
            UIBarButtonSystemItemFlexibleSpace target: nil action: nil];

        // create fixed space for the right spacing
        UIBarButtonItem *fixedSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
            UIBarButtonSystemItemFixedSpace target: nil action: nil];
        fixedSpace.width -= 16.0f;

        // set the toolbar items
        [self.navigationController setToolbarHidden: NO animated: YES];
        [self setToolbarItems: [NSArray arrayWithObjects: flexibleSpace, cancelBarbuttonItem, fixedSpace, nil]
            animated: YES];
    } else if (![[self.reservation isUploaded] boolValue] && ![self.reservation isUploading] && [[self.reservation
        currentStatus] integerValue] == SBReservationStatusFailed) {

        // create a bar button item for the cancelling of the reservation when the reservation is uploaded
        UIButton *cancelReservationButton = [[UIButton alloc] initWithFrame: CGRectMake(160, 0, 160, 44)];
        [cancelReservationButton addTarget: self action: @selector(retryUploadPressed:) forControlEvents:
            UIControlEventTouchUpInside];
        [cancelReservationButton setBackgroundColor: [[SBColorManager defaultColors] redColor]];
        [[cancelReservationButton titleLabel] setFont: [UIFont systemFontOfSize: 13.0f]];
        [cancelReservationButton setTitle: NSLocalizedString(@"overview.viewcontroller.toolbar.retry.upload.title",
            nil) forState: UIControlStateNormal];

        // create the bar button from the other cancel button
        UIBarButtonItem *cancelBarbuttonItem = [[UIBarButtonItem alloc] initWithCustomView: cancelReservationButton]
            ;
        UIBarButtonItem *flexibleSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
            UIBarButtonSystemItemFlexibleSpace target: nil action: nil];

        // create fixed space for the right spacing
        UIBarButtonItem *fixedSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:
            UIBarButtonSystemItemFixedSpace target: nil action: nil];
        fixedSpace.width -= 16.0f;
```

```
        // set the toolbar items
        [self.navigationController setToolbarHidden: NO animated: YES];
        [self setToolbarItems: [NSArray arrayWithObjects: flexibleSpace, cancelBarbuttonItem, fixedSpace, nil]
            animated: YES];
    }
}

/*  When the view will appear on screen set the detail view */
- (void) viewWillAppear:(BOOL)animated
{
    // call super implementation
    [super viewWillAppear: animated];

    // create the date formatter for the title description
    NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
    [dateFormatter setDateFormat: @"EEEE d MMMM"];
    [dateFormatter setLocale: [NSLocale currentLocale]];

    // create the new detaiul view
    SBNavigationBarDetailView *detailView = [[SBNavigationBarDetailView alloc] initWithDetailText: nil];

    // create the attributed string of the following two strings where the date is bold and the time description is
        italic non bold
    NSString *dateDescription = [[dateFormatter stringFromDate: [self.reservation selectedDate]] uppercaseString];
    NSString *timeDescription = [[self.reservation daypart] timeDescription];

    // create the attributed string
    NSMutableAttributedString *timeAttributes = [[NSMutableAttributedString alloc] initWithString: [NSString
        stringWithFormat: @"%@ %@", dateDescription, timeDescription] attributes: @{NSFontAttributeName: [UIFont
        boldSystemFontOfSize: 13.0f]}];

    // add the attributes for the italic string
    [timeAttributes setAttributes: @{NSFontAttributeName: [UIFont italicSystemFontOfSize: 11.0f]} range: NSMakeRange
        ((dateDescription.length + 1), timeDescription.length)];

    // set the title of the detail view with the default used tint color
    [detailView setAttributedTitleString: timeAttributes];

    // set the tint color depending on the navigation type
    if (self.reservationRequiresConfirmation) {
        [detailView setDetailTintColor: [[SBColorManager defaultColors] requestNavigationColor]];
    } else {
        [detailView setDetailTintColor: [[SBColorManager defaultColors] redColor]];
    }

    // set the new detail view
    [(SBNavigationBar *)self.navigationController.navigationBar setDetailView: detailView];
}

/*  When a memory warning is received try to free up resources  */
- (void) didReceiveMemoryWarning
{
    // call super implementation
    [super didReceiveMemoryWarning];
}

/*  When the cancel button was pressed the whole reservation will be cancelled and popped to the root view
    controller   */
- (void) cancelButtonPressed:(UIBarButtonItem *) sender
{
    // create an alert view to approve whether the user really wants to cancel in the final step
    UIAlertView *confirmView = [[UIAlertView alloc] initWithTitle:
        NSLocalizedString(@"overview.viewcontroller.cancel.new.confirm.title", nil) message:
        NSLocalizedString(@"overview.viewcontroller.cancel.new.confirm.description", nil) delegate: self
        cancelButtonTitle: NSLocalizedString(@"overview.viewcontroller.cancel.new.confirm.cancel.title", nil)
        otherButtonTitles: NSLocalizedString(@"overview.viewcontroller.cancel.new.confirm.approve.title", nil), nil]
        ;

    // set the tag and display the alertview
    [confirmView setTag: kSBOverviewControllerAlertViewTagCancelNew];
    [confirmView show];
}

/*  When the final confirmation button was pressed start the uploading process  */
- (void) confirmButtonPressed:(UIBarButtonItem *) sender
{
    // confirm the created reservation
    [[SBReservationManager defaultReservationManager] completeReservationGenerating];

    // set a temporary delegate to the navigation view controller and start to pop to the root view controller
    //[self.navigationController setDelegate: self];
    [self.navigationController popToRootViewControllerAnimated: YES];
}

/*  Returns the size of the detail cell and creates the  */
- (CGFloat) heightForDetails
```

```
{
    // create default x origin coordinate and proceeding object
    CGFloat totalHeight = 0.0f;
    Proceeding *proceeding = [[self.reservation proceedings] anyObject];

    // check whether text has been given, when so add a text label
    if ([[proceeding proceedingDescription] length] > 0 && self.detailLabel == nil) {

        // calculate the rectangle for the given font and text that should be drawn
        CGSize textFrame = [SBViewCompatabilityFactory boundingRectForString: [proceeding proceedingDescription]
            withSize: CGSizeMake(280, 100) options: NSStringDrawingUsesLineFragmentOrigin attributes:
            @{NSFontAttributeName: [UIFont systemFontOfSize: 12.0f]} context: nil];

        // create a label with the provided frame
        self.detailLabel = [[UILabel alloc] initWithFrame: CGRectMake(10, 15, textFrame.width, textFrame.height)];
        [self.detailLabel setFont: [UIFont systemFontOfSize: 12.0f]];
        [self.detailLabel setText: [proceeding proceedingDescription]];
        [self.detailLabel setNumberOfLines: 10];

        // add the text label to the view
        totalHeight += (textFrame.height + 10);
    } else if ([[proceeding proceedingDescription] length] > 0) {

        // the label is already created, append the total height
        totalHeight += (self.detailLabel.frame.size.height + 10);
    }

    // when an image is provided also add it to the cell
    if ([proceeding imageDescriptor] != nil) {

        // add the image in an imageview to the view container
        self.detailImage = [[UIImageView alloc] initWithImage: [[proceeding imageDescriptor] imageThumbnail]];

        // calculate an aspect fit size for the image in the view container
        CGSize imageSize = [self.detailImage.image size];

        // when the image size width is bigger than the frame allows calculate the aspect fit height for the image
            view
        if (imageSize.width > self.informationTableView.frame.size.width) {

            // get the total amount of points that should be removed for a horizontal aspect fit
            CGFloat onePointPercent = 100 / imageSize.width;
            CGFloat totalPercentage = (onePointPercent * self.informationTableView.frame.size.width) / 100;

            // set the new frame of the image view and make the image aspect fit
            [self.detailImage setFrame: CGRectMake(0, totalHeight + 20, self.informationTableView.frame.size.width,
                (imageSize.height * totalPercentage))];
            [self.detailImage setContentMode: UIViewContentModeScaleAspectFit];

        } else {

            // make sure the image view is centered
            [self.detailImage setContentMode: UIViewContentModeCenter];
            [self.detailImage setFrame: CGRectMake(0, totalHeight + 5, self.informationTableView.frame.size.width,
                self.detailImage.image.size.height)];
        }

        // append the total height
        totalHeight += ([self.detailImage frame].size.height + 15);
    } else {

        // append the total height
        totalHeight += 15;
    }

    return totalHeight;
}

/*  Cancels the uploaded reservation when available */
- (void) cancelUploadedReservation:(UIBarButtonItem *) sender
{
    // deactivate the sender
    [sender setEnabled: NO];

    // cancel the reservation object via the reservation manager
    [[SBReservationManager defaultReservationManager] cancelUploadedReservation: self.reservation];
}

/*  When called the reservation object on screen should be refreshed    */
- (void) refreshReservation
{

}

- (void) retryUploadPressed:(UIBarButtonItem *)sender
{
```

```
}

#pragma mark    -
#pragma mark    UITableView delegate & datasource

/*  Returns the amount of sections available in the tableview */
- (NSInteger) numberOfSectionsInTableView:(UITableView *) tableView
{
    // when the detail text or image is set the tableview contains 5 sections, otherwise 4
    if ([[[self.reservation proceedings] anyObject] imageDescriptor] != nil || [[[self.reservation proceedings]
        anyObject] proceedingDescription] length] > 0) {
        return 5;
    } else {
        return 4;
    }
}

/*  Returns the amount of rows in each section, this defaults to 1 for this viewcontroller  */
- (NSInteger) tableView:(UITableView *) tableView numberOfRowsInSection:(NSInteger) section
{
    return 1;
}

/*  Returns the appropriate cell for the corresponding indexPath    */
- (UITableViewCell *) tableView:(UITableView *) tableView cellForRowAtIndexPath:(NSIndexPath *) indexPath
{
    // create the cell
    UITableViewCell *cell = nil;

    // when one of the first 4 cells
    if ([indexPath section] >= 0 && [indexPath section] <= 3) {

        // dequeue on already existing cell when possible
        cell = [tableView dequeueReusableCellWithIdentifier: @"FirstFourCells"];

        // when the request should be confirmated set another color
        if (self.reservationRequiresConfirmation) {
            [(SBSelectableOptionTableViewCell *)cell setLayerTintColor: [[SBColorManager defaultColors]
                requestNavigationColor]];
        } else {
            [(SBSelectableOptionTableViewCell *)cell setLayerTintColor: [[SBColorManager defaultColors] redColor]];
        }

    } else {

        // create a normal cell without using the nibs and reuse identifiers
        cell = [[UITableViewCell alloc] initWithStyle: UITableViewCellStyleDefault reuseIdentifier: nil];
    }


    // retrieve the proceeding object to show details of, set the font that will be used in the cell's textlabel
    Proceeding *proceeding = [[self.reservation proceedings] anyObject];
    [[cell textLabel] setFont: [UIFont boldSystemFontOfSize: 12.0f]];

    // switch to fill the content of the corresponding cell at the given indexpath
    switch ([indexPath section]) {

        case 0:

            // the location of the defect
            [[(SBSelectableOptionTableViewCell *)cell optionLabel] setText: [[[proceeding location]
                locationDescription] uppercaseString]];
            break;

        case 1:

            // the room of the defect
            [[(SBSelectableOptionTableViewCell *)cell optionLabel] setText: [[[proceeding space] spaceDescription]
                uppercaseString]];
            break;

        case 2:

            // the element of the defect
            [[(SBSelectableOptionTableViewCell *)cell optionLabel] setText: [[[proceeding element]
                elementDescription] uppercaseString]];
            break;

        case 3:

            // the actual defect
            [[(SBSelectableOptionTableViewCell *)cell optionLabel] setText: [[[proceeding defect] defectDescription]
                uppercaseString]];
            break;

        case 4:
```

```
            // add the detail text label and image when set
            if (self.detailLabel != nil) {
                [cell.contentView addSubview: self.detailLabel];
            }

            if (self.detailImage != nil) {
                [cell.contentView addSubview: self.detailImage];
            }

            break;
    }

    // set cell selection style to none and return the cell
    [cell setSelectionStyle: UITableViewCellSelectionStyleNone];
    return cell;
}

/*  Returns the height for the header above the given indexpath */
- (CGFloat) tableView:(UITableView *) tableView heightForHeaderInSection:(NSInteger) section
{
    // only the first section header is high
    if (section == 0) {
        return 40;
    } else if (section == 4){
        return 40;
    } else {
        return 4;
    }
}

/*  Returns the height for the footer of the provided section   */
- (CGFloat) tableView:(UITableView *) tableView heightForFooterInSection:(NSInteger) section
{
    // get the total amount of sections in the tableview
    NSInteger totalSectionCount = [self numberOfSectionsInTableView: tableView];

    // the last section gets a higher footer than others
    if (section == (totalSectionCount - 1)) {
        return 30;
    } else {
        return 4;
    }
}

/*  Returns the title for the given section header  */
- (NSString *) tableView:(UITableView *) tableView titleForHeaderInSection:(NSInteger) section
{
    // return the corresponding header title
    if (section == 0) {
        return NSLocalizedString(@"overview.viewcontroller.tableview.subject.title", nil);
    } else if (section == 4) {
        return NSLocalizedString(@"overview.viewcontroller.tableview.remark.title", nil);
    } else {

        // all other sections do not have a title
        return nil;
    }
}

/*  Override the default height of table view cells using this method   */
- (CGFloat) tableView:(UITableView *) tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // append the height of the defect identification cells to 30
    if ([indexPath section] != 4) {
        return 44;
    } else {

        // the row height for the details should be automatically calculated
        return [self heightForDetails];
    }
}

#pragma mark    -
#pragma mark    UIAlertView delegate

/*  When the alertview is dismissed this method is called with the appropriate dismiss index    */
- (void) alertView:(UIAlertView *) alertView willDismissWithButtonIndex:(NSInteger) buttonIndex
{
    // when the cancel button index was not pressed continue, else do nothing
    if (buttonIndex != [alertView cancelButtonIndex]) {

        // when the tag is 1, this means that the user is given the choice to cancel a new created reservation that
            is at the final stage of creation
        if ([alertView tag] == kSBOverviewControllerAlertViewTagCancelNew) {

            // delete the reservation that is created from the CoreData store and pop to the root view controller
            [[SBReservationManager defaultReservationManager] didCancelReservation];
```

```
            [self.navigationController popToRootViewControllerAnimated: YES];

        } else if ([alertView tag] == kSBOVerviewControllerAlertViewTagCancelExisting) {

            // an existing and uploaded reservation should be cancelled
        }
    }
}

#pragma mark    -
#pragma mark    SBOperation delegate

/*  When the operation failed to operate this method is called with the appropriate cancel error    */
- (void) operation:(SBOperation *)operation didFailWithError:(NSError *)error
{

}

- (void) operation:(SBOperation *)operation didFinishWithResult:(SBAPIResponse *)response
{

}

@end
```

```
//
//  SBRoomSelectionViewController.h
//  BouwCloud
//
//  Created by Stephan de Bakker on 21-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBViewController.h"

@interface SBRoomSelectionViewController : SBViewController <UITableViewDataSource, UITableViewDelegate>

// the shosen identifier for location object
@property (nonatomic, strong) NSNumber *locationIdentifier;

// the tableview which holds all room choices
@property (nonatomic, weak) IBOutlet UITableView *roomTableView;

@end
```

```
//
//  SBRoomSelectionViewController.m
//  BouwCloud
//
//  Created by Stephan de Bakker on 21-10-13.
//  Copyright (c) 2013 Stephan de Bakker. All rights reserved.
//

#import "SBRoomSelectionViewController.h"
#import "SBNavigationBar.h"
#import "SBNavigationBarDetailView.h"
#import "SBAppDelegate.h"
#import "SBElementSelectionViewController.h"
#import "SBSelectableOptionTableViewCell.h"
#import "SBReservationManager.h"
#import "SBColorManager.h"

@interface SBRoomSelectionViewController ()

// retrieves the description to use in the navigation bar
- (NSString *) navigationBarDescription;

// fetches all objects that can be selected by the user from the CoreData store
- (void) retrieveSelectableObjects;

// indicates that the view should be reloaded and refresh its content
- (void) refreshContent;
- (void) showActivityView;

// the array of rooms the user can choose from
@property (nonatomic, strong) NSArray *rooms;

// the main thread managed object context
@property (nonatomic, weak) NSManagedObjectContext *mainContext;

// the activity indicator view
@property (nonatomic, strong) UIActivityIndicatorView *activityIndicator;

@end

// the reuse identifier for the table view cell
static NSString *SBRoomSelectionCellReuseIdentifier =   @"SBRoomSelectionReuseIdentifier";

@implementation SBRoomSelectionViewController

/*  Create the view controller from the story board object  */
- (id) initWithCoder:(NSCoder *) aDecoder
{
    self = [super initWithCoder: aDecoder];

    if (self) {

        // set the main threads managed object context
        self.mainContext = [(SBAppDelegate *)[[UIApplication sharedApplication] delegate] managedObjectContext];
    }
    return self;
}

/*  Gets called when the view gets loaded from memory, do view initialization in this method    */
- (void) viewDidLoad
{
    // call super implementation
    [super viewDidLoad];

    // set the title of the view
    [self setTitle: NSLocalizedString(@"location.viewcontroller.title", nil)];

    // set the nib used for reusing cells
    [self.roomTableView registerNib: [UINib nibWithNibName: @"SBSelectableOptionTableViewCell" bundle: [NSBundle
        mainBundle]] forCellReuseIdentifier: SBRoomSelectionCellReuseIdentifier];

    // load the objects
    [self retrieveSelectableObjects];
}

/*  Try to release some memory when called  */
- (void) didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

/*  Set the new detail view and other preparations when the view will come on screen    */
- (void) viewWillAppear:(BOOL)animated
{
    // call super implementatiom
    [super viewWillAppear: animated];
```

```
    // create the detail view for the navigation bar
    SBNavigationBarDetailView *detailView = [[SBNavigationBarDetailView alloc] initWithDetailText: [self
        navigationBarDescription]];
    [detailView setDetailTintColor: [UIColor colorWithRed: 0.54296875 green: 0.8359375 blue: 0.73828125 alpha: 1.0]]
        ;
    [detailView setDetailTitle: NSLocalizedString(@"room.viewcontroller.navigation.detail.title", nil)];
    [detailView setDetailImage: [UIImage imageNamed: @"RuimteStapIndicatie"]];

    // set the new navigation detail view
    [(SBNavigationBar *)self.navigationController.navigationBar setDetailView: detailView];
}

/*  Returns the description to use in the navigation bar    */
- (NSString *) navigationBarDescription
{
    // the title of the navigation detail bar
    NSString *navigationtitle = nil;

    // switch to find the correct string description
    switch ([self.locationIdentifier integerValue]) {
        case 1:

            // set the outside the house description
            navigationtitle = NSLocalizedString(@"room.viewcontroller.navigation.outside.description", nil);
            break;

        case 2:

            // set the inside the house description
            navigationtitle = NSLocalizedString(@"room.viewcontroller.navigation.inside.description", nil);
            break;

        case 3:

            // set the around the house description
            navigationtitle = NSLocalizedString(@"room.viewcontroller.navigation.around.description", nil);
            break;

        case 4:

            // set the general locations description
            navigationtitle = NSLocalizedString(@"room.viewcontroller.navigation.general.description", nil);
            break;

        default:
            break;
    }

    return navigationtitle;
}

/*  Start to load the objects to show as selectable */
- (void) retrieveSelectableObjects
{
    // create the activity indicator to show that the data is being loaded
    [self showActivityView];

    // create a new managed object context to load the data from
    NSManagedObjectContext *managedObjectContext = [[NSManagedObjectContext alloc] initWithConcurrencyType:
        NSPrivateQueueConcurrencyType];
    [managedObjectContext setPersistentStoreCoordinator: [(SBAppDelegate *)[[UIApplication sharedApplication]
        delegate] persistentStoreCoordinator]];

    // perform the block that loads the data
    [managedObjectContext performBlock: ^{

        // create the fetch request for objects to retrieve
        NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] initWithEntityName: @"Space"];
        [fetchRequest setPredicate: [NSPredicate predicateWithFormat: @"location.locationId = %@", self.
            locationIdentifier]];

        // create and add the sort descriptor
        NSSortDescriptor *sortDescriptor = [NSSortDescriptor sortDescriptorWithKey: @"spaceDescription" ascending:
            YES];
        [fetchRequest setSortDescriptors: [NSArray arrayWithObject: sortDescriptor]];

        // create error object and lock the managed object context
        NSError *error = nil;
        [managedObjectContext lock];

        // execute the fetch request
        NSArray *results = [managedObjectContext executeFetchRequest: fetchRequest error: &error];

        // unlock the context
        [managedObjectContext unlock];
```

```
        // when no error occured set the rooms array
        if (error == nil) {

            // create a new array with all object ID's fetched in the background
            NSMutableArray *objectIDs = [NSMutableArray array];

            // add each object ID to the array
            for (NSManagedObject *managedObject in results) {
                [objectIDs addObject: [managedObject objectID]];
            }

            // set the new array and perform the refreshing of the view on the main thread
            self.rooms = [NSArray arrayWithArray: objectIDs];
            [self performSelectorOnMainThread: @selector(refreshContent) withObject: nil waitUntilDone: NO];
        }
    }];
}

/*  Gets called on the main thread and indicates that new data is available to be shown */
- (void) refreshContent
{
    // set the tableview to not be hidden
    [self.roomTableView setHidden: NO];

    // remove the activity indicator when it is added to the view
    if ([self.activityIndicator superview] != nil) {

        // stop animation and remove from memory
        [self.activityIndicator stopAnimating];
        [self.activityIndicator removeFromSuperview];
        self.activityIndicator = nil;
    }

    // reload table data
    [self.roomTableView reloadData];
}

/*  When the data is being loaded from CoreData this method shows the activity indicator    */
- (void) showActivityView
{
    // hide the tableview
    [self.roomTableView setHidden: YES];

    // create and configure the activity indicator view
    self.activityIndicator = [[UIActivityIndicatorView alloc] initWithActivityIndicatorStyle:
        UIActivityIndicatorViewStyleGray];
    [self.activityIndicator setFrame: self.view.bounds];
    [self.activityIndicator setCenter: CGPointMake((self.view.frame.size.width / 2), (self.view.frame.size.height /
        2))];

    // add to the view and start the animation
    [self.view addSubview: self.activityIndicator];
    [self.activityIndicator startAnimating];
}

/*  When the next segue is activated prepare for the next view controller   */
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    // when the element view controller will be shown next set the selected id
    if ([[segue identifier] isEqualToString: @"userDidSelectRoom:"]) {

        // get the index path for the selected row
        NSIndexPath *selectedIndexPath = [self.roomTableView indexPathForSelectedRow];
        NSManagedObject *managedObject = [self.mainContext objectWithID: [self.rooms objectAtIndex:
            [selectedIndexPath section]]];

        // commit the selected room object
        [[SBReservationManager defaultReservationManager] processValue: managedObject
            forCurrentProceedingPropertyKey: @"space"];

        // append the space id to the next view controller
        [[segue destinationViewController] setSpaceIdentifier: [managedObject objectID]];
    }
}

#pragma mark    -
#pragma mark    UITableView delegate & datasource

/*  Defines the amount of sections used in this tableview, each choice represents a section */
- (NSInteger) numberOfSectionsInTableView:(UITableView *) tableView
{
    // return the total amount of rooms
    return [self.rooms count];
}

/*  Returns the amount of rows in each section  */
- (NSInteger) tableView:(UITableView *) tableView numberOfRowsInSection:(NSInteger) section
```

```
{
    // always one row in each section
    return 1;
}

/*  Create and return the table view cell appropriate for the indexpath given   */
- (UITableViewCell *) tableView:(UITableView *) tableView cellForRowAtIndexPath:(NSIndexPath *) indexPath
{
    // dequeue a cell from the reuse stack, there is always a cell available because the nib file is registered
    SBSelectableOptionTableViewCell *cell = (SBSelectableOptionTableViewCell *)[tableView
        dequeueReusableCellWithIdentifier: SBRoomSelectionCellReuseIdentifier];

    // retrieve the object that will be shown in this cell
    NSManagedObject *managedObject = [self.mainContext objectWithID: [self.rooms objectAtIndex: [indexPath section]]
        ];

    // set the text to show in the label
    [[cell optionLabel] setText: [[managedObject valueForKeyPath: @"spaceDescription"] uppercaseString]];
    [cell setLayerTintColor: [[SBColorManager defaultColors] requestNavigationColor]];

    return cell;
}

/*  Returns the height of the header views in the tableview */
- (CGFloat) tableView:(UITableView *) tableView heightForHeaderInSection:(NSInteger) section
{
    // only the first section has a higher header
    if (section == 0) {
        return 20;
    } else {
        return 4;
    }
}

/*  Return the appended footer height for all cells */
- (CGFloat) tableView:(UITableView *) tableView heightForFooterInSection:(NSInteger) section
{
    // only the last cell has a higher footer for the white space
    if (section == ([self.rooms count] - 1)) {
        return 10;
    } else {
        return 4;
    }
}

/*  Gets notified when the user has selected a table view cell, should show the next segue  */
- (void) tableView:(UITableView *) tableView didSelectRowAtIndexPath:(NSIndexPath *) indexPath
{
    // perform the next segue
    [self performSegueWithIdentifier: @"userDidSelectRoom:" sender: self];
}

@end
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB" version="3.0" toolsVersion="4510"
    systemVersion="13A603" targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES"
    initialViewController="T6K-l8-Pc2">
    <dependencies>
        <deployment defaultVersion="1536" identifier="iOS"/>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="3742"/>
    </dependencies>
    <scenes>
        <!--Navigation Controller-->
        <scene sceneID="RYN-Ic-xu1">
            <objects>
                <navigationController definesPresentationContext="YES" id="T6K-l8-Pc2"
                    sceneMemberID="viewController">
                    <extendedEdge key="edgesForExtendedLayout"/>
                    <navigationBar key="navigationBar" contentMode="scaleToFill" id="tKG-60-T8D">
                        <autoresizingMask key="autoresizingMask"/>
                    </navigationBar>
                    <connections>
                        <segue destination="f8x-cJ-580" kind="relationship" relationship="rootViewController" id=
                            "bJe-2n-ejD"/>
                    </connections>
                </navigationController>
                <placeholder placeholderIdentifier="IBFirstResponder" id="5TV-1R-mnT" userLabel="First Responder"
                    sceneMemberID="firstResponder"/>
            </objects>
            <point key="canvasLocation" x="-1106" y="1059"/>
        </scene>
        <!--Home View Controller - Applicatie-->
        <scene sceneID="paC-2x-3jV">
            <objects>
                <viewController storyboardIdentifier="InitialViewController" title="Applicatie" id="f8x-cJ-580"
                    customClass="SBHomeViewController" sceneMemberID="viewController">
                    <layoutGuides>
                        <viewControllerLayoutGuide type="top" id="wk1-4c-QDB"/>
                        <viewControllerLayoutGuide type="bottom" id="hic-bH-RqF"/>
                    </layoutGuides>
                    <view key="view" contentMode="scaleToFill" id="R0w-9h-CP1">
                        <rect key="frame" x="0.0" y="0.0" width="320" height="568"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <subviews>
                            <tableView clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES"
                                dataMode="static" style="grouped" separatorStyle="none" rowHeight="44"
                                sectionHeaderHeight="10" sectionFooterHeight="10"
                                translatesAutoresizingMaskIntoConstraints="NO" id="zgs-Ak-15O">
                                <rect key="frame" x="10" y="0.0" width="300" height="568"/>
                                <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                                <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                                <color key="separatorColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                                <color key="sectionIndexTrackingBackgroundColor" white="0.0" alpha="0.0"
                                    colorSpace="calibratedWhite"/>
                                <sections>
                                    <tableViewSection id="2dg-Xg-4CT">
                                        <cells>
                                            <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                                hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                                indentationWidth="0.0" rowHeight="49" id="2e9-wD-HqU">
                                                <rect key="frame" x="0.0" y="99" width="300" height="49"/>
                                                <autoresizingMask key="autoresizingMask"/>
                                                <tableViewCellContentView key="contentView" opaque="NO"
                                                    clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                                    "center" tableViewCell="2e9-wD-HqU" id="c51-PX-awQ">
                                                    <rect key="frame" x="0.0" y="0.0" width="300" height="49"/>
                                                    <autoresizingMask key="autoresizingMask"/>
                                                </tableViewCellContentView>
                                            </tableViewCell>
                                            <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                                hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                                indentationWidth="0.0" id="96o-xJ-mPh">
                                                <rect key="frame" x="0.0" y="148" width="300" height="44"/>
                                                <autoresizingMask key="autoresizingMask"/>
                                                <tableViewCellContentView key="contentView" opaque="NO"
                                                    clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                                    "center" tableViewCell="96o-xJ-mPh" id="00k-so-ztD">
                                                    <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                                    <autoresizingMask key="autoresizingMask"/>
                                                </tableViewCellContentView>
                                            </tableViewCell>
                                            <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                                hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                                indentationWidth="0.0" id="biD-JY-5aD">
                                                <rect key="frame" x="0.0" y="192" width="300" height="44"/>
                                                <autoresizingMask key="autoresizingMask"/>
                                                <tableViewCellContentView key="contentView" opaque="NO"
                                                    clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                                    "center" tableViewCell="biD-JY-5aD" id="uR7-R1-UP2">
                                                    <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
```

```
                                        <autoresizingMask key="autoresizingMask"/>
                                    </tableViewCellContentView>
                                </tableViewCell>
                            </cells>
                        </tableViewSection>
                    </sections>
                    <connections>
                        <outlet property="dataSource" destination="f8x-cJ-580" id="9G4-1j-XR3"/>
                        <outlet property="delegate" destination="f8x-cJ-580" id="HJH-ML-e6C"/>
                    </connections>
                </tableView>
            </subviews>
            <color key="backgroundColor" white="1" alpha="1" colorSpace="calibratedWhite"/>
            <constraints>
                <constraint firstAttribute="bottom" secondItem="zgs-Ak-15O" secondAttribute="bottom"
                    id="I0I-x0-hIr" userLabel="Vertical Space - View - Table View"/>
                <constraint firstItem="zgs-Ak-15O" firstAttribute="centerX" secondItem="R0w-9h-CP1"
                    secondAttribute="centerX" id="Xx8-4q-6oH"/>
                <constraint firstItem="zgs-Ak-15O" firstAttribute="top" secondItem="R0w-9h-CP1"
                    secondAttribute="top" id="vgQ-R9-bpR"/>
                <constraint firstItem="zgs-Ak-15O" firstAttribute="leading" secondItem="R0w-9h-CP1"
                    secondAttribute="leading" constant="10" id="yKt-dd-gyB"/>
            </constraints>
        </view>
        <navigationItem key="navigationItem" id="wQ5-jB-j2s"/>
        <connections>
            <outlet property="menuTableView" destination="zgs-Ak-15O" id="NWC-x2-kSn"/>
            <segue destination="mHH-eR-a31" kind="push" identifier="userWillEditPersonalData:" id="bW6-
                Om-R8z"/>
            <segue destination="hvx-66-Fv6" kind="push" identifier="userAlreadyHasValidCredentials:"
                id="yWS-qC-4Ad"/>
            <segue destination="a6z-VC-wCy" kind="push" identifier="userDidPressAllReservationCell:"
                id="LA8-Bk-faV"/>
        </connections>
    </viewController>
    <placeholder placeholderIdentifier="IBFirstResponder" id="r6j-p2-eTR" userLabel="First Responder"
        sceneMemberID="firstResponder"/>
</objects>
<point key="canvasLocation" x="-553" y="1059"/>
</scene>
<!--Personal Data View Controller-->
<scene sceneID="vkl-ns-rZq">
    <objects>
        <viewController id="mHH-eR-a31" customClass="SBPersonalDataViewController" sceneMemberID=
            "viewController">
            <layoutGuides>
                <viewControllerLayoutGuide type="top" id="5q8-VF-fme"/>
                <viewControllerLayoutGuide type="bottom" id="BcX-5c-8JB"/>
            </layoutGuides>
            <view key="view" contentMode="scaleToFill" id="fvO-yX-fGh">
                <rect key="frame" x="0.0" y="0.0" width="320" height="524"/>
                <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                <subviews>
                    <tableView clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES"
                        showsVerticalScrollIndicator="NO" dataMode="static" style="grouped" separatorStyle=
                        "none" rowHeight="44" sectionHeaderHeight="10" sectionFooterHeight="10"
                        translatesAutoresizingMaskIntoConstraints="NO" id="4XD-9k-IAH">
                        <rect key="frame" x="10" y="0.0" width="300" height="524"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                        <color key="separatorColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                        <sections>
                            <tableViewSection id="GFE-67-OKA">
                                <cells>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="SHf-ix-FUI">
                                        <rect key="frame" x="0.0" y="99" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="SHf-ix-FUI" id="ngi-70-udr">
                                            <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                        </tableViewCellContentView>
                                    </tableViewCell>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="tig-OU-e89">
                                        <rect key="frame" x="0.0" y="143" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="tig-OU-e89" id="Q62-L7-izS">
                                            <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                        </tableViewCellContentView>
```

```
                                    </tableViewCell>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="Qoy-EZ-pEr">
                                        <rect key="frame" x="0.0" y="187" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="Qoy-EZ-pEr" id="ead-sV-xgS">
                                            <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                        </tableViewCellContentView>
                                    </tableViewCell>
                                </cells>
                            </tableViewSection>
                        </sections>
                        <connections>
                            <outlet property="dataSource" destination="mHH-eR-a31" id="Ys5-YI-Xby"/>
                            <outlet property="delegate" destination="mHH-eR-a31" id="Ot0-zB-pOy"/>
                        </connections>
                    </tableView>
                </subviews>
                <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace=
                    "calibratedWhite"/>
                <constraints>
                    <constraint firstAttribute="bottom" secondItem="4XD-9k-IAH" secondAttribute="bottom"
                        id="0TK-gZ-4Gb"/>
                    <constraint firstAttribute="trailing" secondItem="4XD-9k-IAH" secondAttribute="trailing"
                        constant="10" id="kXg-2v-3j7"/>
                    <constraint firstItem="4XD-9k-IAH" firstAttribute="top" secondItem="fvO-yX-fGh"
                        secondAttribute="top" id="yX4-zu-78F"/>
                    <constraint firstItem="4XD-9k-IAH" firstAttribute="leading" secondItem="fvO-yX-fGh"
                        secondAttribute="leading" constant="10" id="ymQ-7A-Hbs"/>
                </constraints>
            </view>
            <toolbarItems>
                <barButtonItem style="plain" id="yzf-4s-D6c">
                    <inset key="imageInsets" minX="0.0" minY="-3" maxX="0.0" maxY="0.0"/>
                    <color key="tintColor" white="0.66666666666666663" alpha="1"
                        colorSpace="calibratedWhite"/>
                    <userDefinedRuntimeAttributes>
                        <userDefinedRuntimeAttribute type="string" keyPath="title" value="ANNULEREN"
                            localized="YES"/>
                    </userDefinedRuntimeAttributes>
                    <connections>
                        <action selector="cancelButtonPressed:" destination="mHH-eR-a31" id="PL1-lM-FC3"/>
                    </connections>
                </barButtonItem>
                <barButtonItem style="plain" systemItem="flexibleSpace" id="vtR-wb-Mw1"/>
            </toolbarItems>
            <navigationItem key="navigationItem" id="Jcn-3U-Dl7"/>
            <simulatedToolbarMetrics key="simulatedBottomBarMetrics" translucent="NO"/>
            <connections>
                <outlet property="tableView" destination="4XD-9k-IAH" id="wAa-E8-cqq"/>
                <segue destination="hvx-66-Fv6" kind="push" identifier="userDidEnterValidCredentials:" id=
                    "xxn-gz-vEJ"/>
            </connections>
        </viewController>
        <placeholder placeholderIdentifier="IBFirstResponder" id="kSH-rt-YSW" userLabel="First Responder"
            sceneMemberID="firstResponder"/>
    </objects>
    <point key="canvasLocation" x="107" y="1609"/>
</scene>
<!--All Reservations View Controller-->
<scene sceneID="W48-ff-Wsh">
    <objects>
        <viewController id="a6z-VC-wCy" customClass="SBAllReservationsViewController" sceneMemberID=
            "viewController">
            <layoutGuides>
                <viewControllerLayoutGuide type="top" id="gUv-kx-bhP"/>
                <viewControllerLayoutGuide type="bottom" id="rK2-qk-nEA"/>
            </layoutGuides>
            <view key="view" contentMode="scaleToFill" id="6Tf-RM-rkz">
                <rect key="frame" x="0.0" y="0.0" width="320" height="568"/>
                <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                <subviews>
                    <tableView clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES"
                        showsVerticalScrollIndicator="NO" dataMode="static" style="grouped" separatorStyle=
                        "none" rowHeight="44" sectionHeaderHeight="10" sectionFooterHeight="10"
                        translatesAutoresizingMaskIntoConstraints="NO" id="LKs-pF-2kd">
                        <rect key="frame" x="10" y="0.0" width="300" height="568"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                        <sections>
                            <tableViewSection id="o6P-29-5Eo">
                                <cells>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
```

```
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="ERu-1g-iMs">
                                        <rect key="frame" x="0.0" y="99" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="ERu-1g-iMs" id="Mpg-6y-ANh">
                                            <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                        </tableViewCellContentView>
                                    </tableViewCell>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="BSa-tB-h93">
                                        <rect key="frame" x="0.0" y="143" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="BSa-tB-h93" id="RTu-rd-3TV">
                                            <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                        </tableViewCellContentView>
                                    </tableViewCell>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="LfM-UF-X90">
                                        <rect key="frame" x="0.0" y="187" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="LfM-UF-X90" id="JkX-Kf-F4c">
                                            <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                        </tableViewCellContentView>
                                    </tableViewCell>
                                </cells>
                            </tableViewSection>
                        </sections>
                        <connections>
                            <outlet property="dataSource" destination="a6z-VC-wCy" id="ViM-UH-Rd0"/>
                            <outlet property="delegate" destination="a6z-VC-wCy" id="6xJ-06-Wb7"/>
                        </connections>
                    </tableView>
                </subviews>
                <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace=
                    "calibratedWhite"/>
                <constraints>
                    <constraint firstAttribute="trailing" secondItem="LKs-pF-2kd" secondAttribute="trailing"
                        constant="10" id="0Yq-wD-v9s"/>
                    <constraint firstItem="LKs-pF-2kd" firstAttribute="top" secondItem="6Tf-RM-rkz"
                        secondAttribute="top" id="GcV-wf-wRM"/>
                    <constraint firstAttribute="bottom" secondItem="LKs-pF-2kd" secondAttribute="bottom"
                        id="JPj-bS-MAu"/>
                    <constraint firstItem="LKs-pF-2kd" firstAttribute="leading" secondItem="6Tf-RM-rkz"
                        secondAttribute="leading" constant="10" id="jKy-on-Ee9"/>
                </constraints>
            </view>
            <navigationItem key="navigationItem" id="lzf-4S-gWH"/>
            <connections>
                <outlet property="reservationTableView" destination="LKs-pF-2kd" id="aOZ-HA-zVS"/>
                <segue destination="gjR-P9-utp" kind="push" identifier="userWantsReservationInfo:" id="yNR-
                    X8-qbg"/>
            </connections>
        </viewController>
        <placeholder placeholderIdentifier="IBFirstResponder" id="pcZ-tF-Prq" userLabel="First Responder"
            sceneMemberID="firstResponder"/>
    </objects>
    <point key="canvasLocation" x="107" y="555"/>
</scene>
<!--Location Selection View Controller-->
<scene sceneID="jex-8x-AJt">
    <objects>
        <viewController id="hvx-66-Fv6" customClass="SBLocationSelectionViewController" sceneMemberID=
            "viewController">
            <layoutGuides>
                <viewControllerLayoutGuide type="top" id="cgO-aF-LZf"/>
                <viewControllerLayoutGuide type="bottom" id="hUV-nq-82a"/>
            </layoutGuides>
            <view key="view" contentMode="scaleToFill" id="F4s-XN-o3p">
                <rect key="frame" x="0.0" y="0.0" width="320" height="524"/>
                <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                <subviews>
                    <collectionView opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                        "scaleToFill" fixedFrame="YES" minimumZoomScale="0.0" maximumZoomScale="0.0"
                        dataMode="prototypes" translatesAutoresizingMaskIntoConstraints="NO" id="uVi-Km-
                        kFk">
                        <rect key="frame" x="0.0" y="0.0" width="320" height="524"/>
```

```
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                        <collectionViewFlowLayout key="collectionViewLayout" minimumLineSpacing="10"
                            minimumInteritemSpacing="10" id="asI-g2-V4e">
                            <size key="itemSize" width="50" height="50"/>
                            <size key="headerReferenceSize" width="0.0" height="0.0"/>
                            <size key="footerReferenceSize" width="0.0" height="0.0"/>
                            <inset key="sectionInset" minX="0.0" minY="0.0" maxX="0.0" maxY="0.0"/>
                        </collectionViewFlowLayout>
                        <cells>
                            <collectionViewCell opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES"
                                contentMode="center" id="INv-PN-Kcc">
                                <rect key="frame" x="0.0" y="64" width="50" height="50"/>
                                <autoresizingMask key="autoresizingMask"/>
                                <view key="contentView" opaque="NO" clipsSubviews="YES"
                                    multipleTouchEnabled="YES" contentMode="center">
                                    <rect key="frame" x="0.0" y="0.0" width="50" height="50"/>
                                    <autoresizingMask key="autoresizingMask"/>
                                    <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace=
                                        "calibratedWhite"/>
                                </view>
                            </collectionViewCell>
                        </cells>
                        <connections>
                            <outlet property="dataSource" destination="hvx-66-Fv6" id="n0P-SZ-raz"/>
                            <outlet property="delegate" destination="hvx-66-Fv6" id="TWb-Co-egl"/>
                        </connections>
                    </collectionView>
                </subviews>
                <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace=
                    "calibratedWhite"/>
            </view>
            <navigationItem key="navigationItem" id="6ft-Xi-bXc"/>
            <connections>
                <outlet property="locationCollectionView" destination="uVi-Km-kFk" id="neD-at-jFu"/>
                <segue destination="tRP-Y8-TEC" kind="push" identifier="userDidSelectLocation:" id="L94-Wp-
                    Udl"/>
            </connections>
        </viewController>
        <placeholder placeholderIdentifier="IBFirstResponder" id="4LU-3g-Qjp" userLabel="First Responder"
            sceneMemberID="firstResponder"/>
    </objects>
    <point key="canvasLocation" x="759" y="1059"/>
</scene>
<!--Room Selection View Controller-->
<scene sceneID="BxU-3R-Tqr">
    <objects>
        <viewController id="tRP-Y8-TEC" customClass="SBRoomSelectionViewController" sceneMemberID=
            "viewController">
            <layoutGuides>
                <viewControllerLayoutGuide type="top" id="Rur-iy-p1M"/>
                <viewControllerLayoutGuide type="bottom" id="7E5-ZS-I8w"/>
            </layoutGuides>
            <view key="view" contentMode="scaleToFill" id="eim-jz-CFq">
                <rect key="frame" x="0.0" y="0.0" width="320" height="568"/>
                <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                <subviews>
                    <tableView clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES"
                        showsVerticalScrollIndicator="NO" dataMode="static" style="grouped" separatorStyle=
                        "none" rowHeight="44" sectionHeaderHeight="10" sectionFooterHeight="10"
                        translatesAutoresizingMaskIntoConstraints="NO" id="4Ex-nS-qDa">
                        <rect key="frame" x="10" y="0.0" width="300" height="568"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                        <sections>
                            <tableViewSection id="Tzh-Ih-u1D">
                                <cells>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="5uR-15-t0W">
                                        <rect key="frame" x="0.0" y="99" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="5uR-15-t0W" id="fwQ-jH-84e">
                                            <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                        </tableViewCellContentView>
                                    </tableViewCell>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="cpK-ir-qUN">
                                        <rect key="frame" x="0.0" y="143" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="cpK-ir-qUN" id="EdD-D1-kLy">
```

```
                                        <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                    </tableViewCellContentView>
                                </tableViewCell>
                                <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                    hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                    indentationWidth="0.0" id="XTe-RB-cux">
                                    <rect key="frame" x="0.0" y="187" width="300" height="44"/>
                                    <autoresizingMask key="autoresizingMask"/>
                                    <tableViewCellContentView key="contentView" opaque="NO"
                                        clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                        "center" tableViewCell="XTe-RB-cux" id="s3E-gP-GSK">
                                        <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                    </tableViewCellContentView>
                                </tableViewCell>
                            </cells>
                        </tableViewSection>
                    </sections>
                    <connections>
                        <outlet property="dataSource" destination="tRP-Y8-TEC" id="CDg-bX-ryO"/>
                        <outlet property="delegate" destination="tRP-Y8-TEC" id="ZHI-qQ-MnV"/>
                    </connections>
                </tableView>
            </subviews>
            <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace=
                "calibratedWhite"/>
            <constraints>
                <constraint firstItem="4Ex-nS-qDa" firstAttribute="leading" secondItem="eim-jz-CFq"
                    secondAttribute="leading" constant="10" id="9l1-ta-Phe"/>
                <constraint firstItem="4Ex-nS-qDa" firstAttribute="top" secondItem="eim-jz-CFq"
                    secondAttribute="top" id="eV9-Pd-v50"/>
                <constraint firstAttribute="trailing" secondItem="4Ex-nS-qDa" secondAttribute="trailing"
                    constant="10" id="gyv-jW-zjl"/>
                <constraint firstAttribute="bottom" secondItem="4Ex-nS-qDa" secondAttribute="bottom"
                    id="rhu-Rx-XQy"/>
            </constraints>
        </view>
        <extendedEdge key="edgesForExtendedLayout" top="YES"/>
        <navigationItem key="navigationItem" id="5yv-Z2-MRa"/>
        <nil key="simulatedBottomBarMetrics"/>
        <connections>
            <outlet property="roomTableView" destination="4Ex-nS-qDa" id="5K8-ch-gpb"/>
            <segue destination="GwY-Mi-QGb" kind="push" identifier="userDidSelectRoom:" id="imd-Qd-MRh"/
                >
        </connections>
    </viewController>
    <placeholder placeholderIdentifier="IBFirstResponder" id="TYb-hY-oMn" userLabel="First Responder"
        sceneMemberID="firstResponder"/>
</objects>
<point key="canvasLocation" x="1298" y="1059"/>
</scene>
<!--Reservation Overview View Controller-->
<scene sceneID="lOs-gR-v56">
    <objects>
        <viewController id="gjR-P9-utp" customClass="SBReservationOverviewViewController" sceneMemberID=
            "viewController">
            <layoutGuides>
                <viewControllerLayoutGuide type="top" id="fGl-Yp-fcB"/>
                <viewControllerLayoutGuide type="bottom" id="Dqd-8y-V2q"/>
            </layoutGuides>
            <view key="view" contentMode="scaleToFill" id="wj9-gB-9Cy">
                <rect key="frame" x="0.0" y="0.0" width="320" height="524"/>
                <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                <subviews>
                    <tableView clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES"
                        showsVerticalScrollIndicator="NO" dataMode="prototypes" style="grouped"
                        separatorStyle="none" rowHeight="44" sectionHeaderHeight="10"
                        sectionFooterHeight="10" translatesAutoresizingMaskIntoConstraints="NO" id="E5d-WD-
                        CsC">
                        <rect key="frame" x="10" y="0.0" width="300" height="524"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                        <prototypes>
                            <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                hidesAccessoryWhenEditing="NO" indentationLevel="1" indentationWidth="0.0"
                                reuseIdentifier="FirstFourCells" id="fQv-0r-WIK" customClass=
                                "SBSelectableOptionTableViewCell">
                                <rect key="frame" x="0.0" y="119" width="300" height="44"/>
                                <autoresizingMask key="autoresizingMask"/>
                                <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES"
                                    multipleTouchEnabled="YES" contentMode="center" tableViewCell="fQv-0r-
                                    WIK" id="Rad-9N-Spf">
                                    <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                    <autoresizingMask key="autoresizingMask"/>
                                </tableViewCellContentView>
                            </tableViewCell>
```

```
                                <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                    hidesAccessoryWhenEditing="NO" indentationLevel="1" indentationWidth="0.0"
                                    reuseIdentifier="FirstFourCells" id="nXM-fF-iPS" customClass=
                                    "SBSelectableOptionTableViewCell">
                                    <rect key="frame" x="0.0" y="163" width="300" height="44"/>
                                    <autoresizingMask key="autoresizingMask"/>
                                    <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES"
                                        multipleTouchEnabled="YES" contentMode="center" tableViewCell="nXM-fF-
                                        iPS" id="egm-sX-n0b">
                                        <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                    </tableViewCellContentView>
                                </tableViewCell>
                                <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                    hidesAccessoryWhenEditing="NO" indentationLevel="1" indentationWidth="0.0"
                                    reuseIdentifier="FirstFourCells" id="n42-mx-aa2" customClass=
                                    "SBSelectableOptionTableViewCell">
                                    <rect key="frame" x="0.0" y="207" width="300" height="44"/>
                                    <autoresizingMask key="autoresizingMask"/>
                                    <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES"
                                        multipleTouchEnabled="YES" contentMode="center" tableViewCell="n42-mx-
                                        aa2" id="nod-x0-S3i">
                                        <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                    </tableViewCellContentView>
                                </tableViewCell>
                                <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                    hidesAccessoryWhenEditing="NO" indentationLevel="1" indentationWidth="0.0"
                                    reuseIdentifier="FirstFourCells" id="7QW-KC-h4J" customClass=
                                    "SBSelectableOptionTableViewCell">
                                    <rect key="frame" x="0.0" y="251" width="300" height="44"/>
                                    <autoresizingMask key="autoresizingMask"/>
                                    <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES"
                                        multipleTouchEnabled="YES" contentMode="center" tableViewCell="7QW-KC-
                                        h4J" id="Ac8-xW-QTo">
                                        <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                    </tableViewCellContentView>
                                </tableViewCell>
                            </prototypes>
                            <sections/>
                            <connections>
                                <outlet property="dataSource" destination="gjR-P9-utp" id="xV0-aV-Ner"/>
                                <outlet property="delegate" destination="gjR-P9-utp" id="hj0-GI-lue"/>
                            </connections>
                        </tableView>
                    </subviews>
                    <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace=
                        "calibratedWhite"/>
                    <constraints>
                        <constraint firstItem="E5d-WD-CsC" firstAttribute="leading" secondItem="wj9-gB-9Cy"
                            secondAttribute="leading" constant="10" id="5KH-u2-Tjk"/>
                        <constraint firstAttribute="trailing" secondItem="E5d-WD-CsC" secondAttribute="trailing"
                            constant="10" id="Cro-1S-Lj8"/>
                        <constraint firstAttribute="bottom" secondItem="E5d-WD-CsC" secondAttribute="bottom"
                            id="gjr-cN-EDA"/>
                        <constraint firstItem="E5d-WD-CsC" firstAttribute="top" secondItem="wj9-gB-9Cy"
                            secondAttribute="top" id="v9j-b1-HJW"/>
                    </constraints>
                </view>
                <toolbarItems/>
                <navigationItem key="navigationItem" id="zP1-Vq-DOh"/>
                <connections>
                    <outlet property="informationTableView" destination="E5d-WD-CsC" id="aNX-Pr-G2S"/>
                </connections>
            </viewController>
            <placeholder placeholderIdentifier="IBFirstResponder" id="RMt-l8-MTn" userLabel="First Responder"
                sceneMemberID="firstResponder"/>
        </objects>
        <point key="canvasLocation" x="4045" y="1059"/>
    </scene>
    <!--Element Selection View Controller-->
    <scene sceneID="q4l-wL-a3N">
        <objects>
            <viewController id="GwY-Mi-QGb" customClass="SBElementSelectionViewController" sceneMemberID=
                "viewController">
                <layoutGuides>
                    <viewControllerLayoutGuide type="top" id="eux-oP-CWf"/>
                    <viewControllerLayoutGuide type="bottom" id="8mx-io-OFQ"/>
                </layoutGuides>
                <view key="view" contentMode="scaleToFill" id="dIP-qr-ySh">
                    <rect key="frame" x="0.0" y="0.0" width="320" height="568"/>
                    <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                    <subviews>
                        <tableView clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES"
                            showsVerticalScrollIndicator="NO" dataMode="static" style="grouped" separatorStyle=
                            "none" rowHeight="44" sectionHeaderHeight="10" sectionFooterHeight="10"
```

```
                        translatesAutoresizingMaskIntoConstraints="NO" id="ugZ-X6-Gze">
                        <rect key="frame" x="10" y="0.0" width="300" height="568"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                        <sections>
                            <tableViewSection id="A81-zo-wn9">
                                <cells>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="q8e-Qq-ZPy">
                                        <rect key="frame" x="0.0" y="99" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="q8e-Qq-ZPy" id="8na-pF-4C8">
                                            <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                        </tableViewCellContentView>
                                    </tableViewCell>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="RGP-x7-sjb">
                                        <rect key="frame" x="0.0" y="143" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="RGP-x7-sjb" id="lTQ-eO-AZc">
                                            <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                        </tableViewCellContentView>
                                    </tableViewCell>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="9Es-eC-q3K">
                                        <rect key="frame" x="0.0" y="187" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="9Es-eC-q3K" id="X5D-px-WK0">
                                            <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                        </tableViewCellContentView>
                                    </tableViewCell>
                                </cells>
                            </tableViewSection>
                        </sections>
                        <connections>
                            <outlet property="dataSource" destination="GwY-Mi-QGb" id="FSG-6h-vbp"/>
                            <outlet property="delegate" destination="GwY-Mi-QGb" id="FoD-ZC-xYY"/>
                        </connections>
                    </tableView>
                </subviews>
                <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace=
                    "calibratedWhite"/>
                <constraints>
                    <constraint firstAttribute="bottom" secondItem="ugZ-X6-Gze" secondAttribute="bottom"
                        id="LkT-29-vJN"/>
                    <constraint firstItem="ugZ-X6-Gze" firstAttribute="leading" secondItem="dIP-qr-ySh"
                        secondAttribute="leading" constant="10" id="cQT-Vv-AZh"/>
                    <constraint firstAttribute="trailing" secondItem="ugZ-X6-Gze" secondAttribute="trailing"
                        constant="10" id="fcw-LK-QaD"/>
                    <constraint firstItem="ugZ-X6-Gze" firstAttribute="top" secondItem="dIP-qr-ySh"
                        secondAttribute="top" id="jiT-YA-xkA"/>
                </constraints>
            </view>
            <navigationItem key="navigationItem" id="FDY-Yf-doo"/>
            <connections>
                <outlet property="elementTableView" destination="ugZ-X6-Gze" id="6XN-0h-xco"/>
                <segue destination="Qy4-st-ZrM" kind="push" identifier="userDidSelectElement:" id="HKl-x4-
                    c5X"/>
            </connections>
        </viewController>
        <placeholder placeholderIdentifier="IBFirstResponder" id="fHC-Eh-bNc" userLabel="First Responder"
            sceneMemberID="firstResponder"/>
    </objects>
    <point key="canvasLocation" x="1904" y="1059"/>
</scene>
<!--Defect Selection View Controller-->
<scene sceneID="45w-bs-33U">
    <objects>
        <viewController id="Qy4-st-ZrM" customClass="SBDefectSelectionViewController" sceneMemberID=
            "viewController">
            <layoutGuides>
                <viewControllerLayoutGuide type="top" id="WUf-Lr-6XI"/>
                <viewControllerLayoutGuide type="bottom" id="5rd-38-I8V"/>
            </layoutGuides>
            <view key="view" contentMode="scaleToFill" id="BmS-Qh-95T">
```

```
                    <rect key="frame" x="0.0" y="0.0" width="320" height="568"/>
                    <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                    <subviews>
                        <tableView clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES"
                            showsVerticalScrollIndicator="NO" dataMode="static" style="grouped" separatorStyle=
                            "none" rowHeight="44" sectionHeaderHeight="10" sectionFooterHeight="10"
                            translatesAutoresizingMaskIntoConstraints="NO" id="Jcf-NW-hxD">
                            <rect key="frame" x="10" y="0.0" width="300" height="568"/>
                            <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                            <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                            <sections>
                                <tableViewSection id="UqS-NG-rm4">
                                    <cells>
                                        <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                            hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                            indentationWidth="0.0" id="wey-LM-a6U">
                                            <rect key="frame" x="0.0" y="99" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                            <tableViewCellContentView key="contentView" opaque="NO"
                                                clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                                "center" tableViewCell="wey-LM-a6U" id="SB1-Oi-lEF">
                                                <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                                <autoresizingMask key="autoresizingMask"/>
                                            </tableViewCellContentView>
                                        </tableViewCell>
                                        <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                            hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                            indentationWidth="0.0" id="O2K-6U-cas">
                                            <rect key="frame" x="0.0" y="143" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                            <tableViewCellContentView key="contentView" opaque="NO"
                                                clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                                "center" tableViewCell="O2K-6U-cas" id="981-nU-gGk">
                                                <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                                <autoresizingMask key="autoresizingMask"/>
                                            </tableViewCellContentView>
                                        </tableViewCell>
                                        <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                            hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                            indentationWidth="0.0" id="w22-Wh-UKl">
                                            <rect key="frame" x="0.0" y="187" width="300" height="44"/>
                                            <autoresizingMask key="autoresizingMask"/>
                                            <tableViewCellContentView key="contentView" opaque="NO"
                                                clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                                "center" tableViewCell="w22-Wh-UKl" id="n9q-XM-P2F">
                                                <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                                <autoresizingMask key="autoresizingMask"/>
                                            </tableViewCellContentView>
                                        </tableViewCell>
                                    </cells>
                                </tableViewSection>
                            </sections>
                            <connections>
                                <outlet property="dataSource" destination="Qy4-st-ZrM" id="hjw-bh-qoe"/>
                                <outlet property="delegate" destination="Qy4-st-ZrM" id="Owf-fW-vdX"/>
                            </connections>
                        </tableView>
                    </subviews>
                    <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace=
                        "calibratedWhite"/>
                    <constraints>
                        <constraint firstItem="Jcf-NW-hxD" firstAttribute="top" secondItem="BmS-Qh-95T"
                            secondAttribute="top" id="Hl6-Cc-mis"/>
                        <constraint firstAttribute="trailing" secondItem="Jcf-NW-hxD" secondAttribute="trailing"
                            constant="10" id="JTD-Lk-WFa"/>
                        <constraint firstItem="Jcf-NW-hxD" firstAttribute="leading" secondItem="BmS-Qh-95T"
                            secondAttribute="leading" constant="10" id="UHA-zT-P95"/>
                        <constraint firstAttribute="bottom" secondItem="Jcf-NW-hxD" secondAttribute="bottom"
                            id="g48-xs-mYG"/>
                    </constraints>
                </view>
                <navigationItem key="navigationItem" id="nQr-Os-8zy"/>
                <connections>
                    <outlet property="defectTableView" destination="Jcf-NW-hxD" id="hZW-r1-gPI"/>
                    <segue destination="l5f-lL-8y5" kind="push" identifier="userDidSelectDefect:" id="5j1-rG-
                        L93"/>
                </connections>
            </viewController>
            <placeholder placeholderIdentifier="IBFirstResponder" id="7VS-O0-nHu" userLabel="First Responder"
                sceneMemberID="firstResponder"/>
        </objects>
        <point key="canvasLocation" x="2453" y="1059"/>
    </scene>
    <!--Optional Information View Controller-->
    <scene sceneID="Skk-dS-HZA">
        <objects>
            <viewController id="l5f-lL-8y5" customClass="SBOptionalInformationViewController" sceneMemberID=
```

```
            "viewController">
            <layoutGuides>
                <viewControllerLayoutGuide type="top" id="eZK-Xo-hGn"/>
                <viewControllerLayoutGuide type="bottom" id="EPn-9B-WEs"/>
            </layoutGuides>
            <view key="view" contentMode="scaleToFill" id="USw-x0-Jc0">
                <rect key="frame" x="0.0" y="0.0" width="320" height="524"/>
                <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                <subviews>
                    <scrollView clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="scaleToFill"
                        showsVerticalScrollIndicator="NO" translatesAutoresizingMaskIntoConstraints="NO"
                        id="3cc-60-ZF8">
                        <rect key="frame" x="0.0" y="0.0" width="320" height="524"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                    </scrollView>
                </subviews>
                <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace=
                    "calibratedWhite"/>
                <constraints>
                    <constraint firstAttribute="bottom" secondItem="3cc-60-ZF8" secondAttribute="bottom"
                        id="K2m-0W-6Ja"/>
                    <constraint firstItem="3cc-60-ZF8" firstAttribute="top" secondItem="USw-x0-Jc0"
                        secondAttribute="top" id="LuE-dN-yIw"/>
                    <constraint firstItem="3cc-60-ZF8" firstAttribute="leading" secondItem="USw-x0-Jc0"
                        secondAttribute="leading" id="dtS-bL-IrU"/>
                    <constraint firstAttribute="trailing" secondItem="3cc-60-ZF8" secondAttribute="trailing"
                        id="o6o-e3-OGQ"/>
                </constraints>
            </view>
            <toolbarItems>
                <barButtonItem image="CameraButton.png" id="PA4-rB-aj2">
                    <color key="tintColor" red="0.54353900547445255" green="0.83938754562043794"
                        blue="0.74230155109489049" alpha="1" colorSpace="calibratedRGB"/>
                    <connections>
                        <action selector="cameraBarButtonItemPressed:" destination="l5f-lL-8y5" id="BXr-k9-
                            Tyb"/>
                    </connections>
                </barButtonItem>
                <barButtonItem style="plain" systemItem="flexibleSpace" id="AVQ-s8-pHf"/>
            </toolbarItems>
            <navigationItem key="navigationItem" id="SU1-ql-FYv"/>
            <simulatedToolbarMetrics key="simulatedBottomBarMetrics" translucent="NO"/>
            <connections>
                <outlet property="contentScrollView" destination="3cc-60-ZF8" id="3Ri-Vn-kWm"/>
                <segue destination="SPh-9W-vhz" kind="push" identifier="userDidFinishOptionalOptions:" id=
                    "kpi-11-oKi"/>
            </connections>
        </viewController>
        <placeholder placeholderIdentifier="IBFirstResponder" id="K7O-wR-BwW" userLabel="First Responder"
            sceneMemberID="firstResponder"/>
    </objects>
    <point key="canvasLocation" x="2988" y="1059"/>
</scene>
<!--Appointment View Controller-->
<scene sceneID="qh2-3c-SOm">
    <objects>
        <viewController id="SPh-9W-vhz" customClass="SBAppointmentViewController" sceneMemberID=
            "viewController">
            <layoutGuides>
                <viewControllerLayoutGuide type="top" id="8hF-r5-xyy"/>
                <viewControllerLayoutGuide type="bottom" id="wIb-DH-Zw0"/>
            </layoutGuides>
            <view key="view" contentMode="scaleToFill" id="UhV-7E-psq">
                <rect key="frame" x="0.0" y="0.0" width="320" height="524"/>
                <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                <subviews>
                    <tableView clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES"
                        scrollEnabled="NO" showsVerticalScrollIndicator="NO" dataMode="static" style=
                        "grouped" separatorStyle="none" rowHeight="44" sectionHeaderHeight="10"
                        sectionFooterHeight="10" translatesAutoresizingMaskIntoConstraints="NO" id="5Bb-
                        Q9-0Xd">
                        <rect key="frame" x="10" y="0.0" width="300" height="524"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                        <color key="separatorColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
                        <sections>
                            <tableViewSection id="n7O-Wt-CpV">
                                <cells>
                                    <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                        hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                        indentationWidth="0.0" id="IXX-zV-AVL">
                                        <rect key="frame" x="0.0" y="99" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                        <tableViewCellContentView key="contentView" opaque="NO"
                                            clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                            "center" tableViewCell="IXX-zV-AVL" id="tOM-Os-VVi">
```

```
                                        <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                    </tableViewCellContentView>
                                </tableViewCell>
                                <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                    hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                    indentationWidth="0.0" id="qM8-ES-evo">
                                    <rect key="frame" x="0.0" y="143" width="300" height="44"/>
                                    <autoresizingMask key="autoresizingMask"/>
                                    <tableViewCellContentView key="contentView" opaque="NO"
                                        clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                        "center" tableViewCell="qM8-ES-evo" id="VFY-qa-ge0">
                                        <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                    </tableViewCellContentView>
                                </tableViewCell>
                                <tableViewCell contentMode="scaleToFill" selectionStyle="blue"
                                    hidesAccessoryWhenEditing="NO" indentationLevel="1"
                                    indentationWidth="0.0" id="NYc-Ow-Gyr">
                                    <rect key="frame" x="0.0" y="187" width="300" height="44"/>
                                    <autoresizingMask key="autoresizingMask"/>
                                    <tableViewCellContentView key="contentView" opaque="NO"
                                        clipsSubviews="YES" multipleTouchEnabled="YES" contentMode=
                                        "center" tableViewCell="NYc-Ow-Gyr" id="Tee-uE-dtG">
                                        <rect key="frame" x="0.0" y="0.0" width="300" height="44"/>
                                        <autoresizingMask key="autoresizingMask"/>
                                    </tableViewCellContentView>
                                </tableViewCell>
                            </cells>
                        </tableViewSection>
                    </sections>
                    <connections>
                        <outlet property="dataSource" destination="SPh-9W-vhz" id="GJ4-m3-IaN"/>
                        <outlet property="delegate" destination="SPh-9W-vhz" id="UUc-Yc-B9u"/>
                    </connections>
                </tableView>
            </subviews>
            <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace=
                "calibratedWhite"/>
            <constraints>
                <constraint firstItem="5Bb-Q9-0Xd" firstAttribute="top" secondItem="UhV-7E-psq"
                    secondAttribute="top" id="0M5-65-Hmj"/>
                <constraint firstItem="5Bb-Q9-0Xd" firstAttribute="leading" secondItem="UhV-7E-psq"
                    secondAttribute="leading" constant="10" id="VdE-NX-Qac"/>
                <constraint firstAttribute="trailing" secondItem="5Bb-Q9-0Xd" secondAttribute="trailing"
                    constant="10" id="vUe-vK-1Yv"/>
                <constraint firstAttribute="bottom" secondItem="5Bb-Q9-0Xd" secondAttribute="bottom"
                    id="wVY-OC-STB"/>
            </constraints>
        </view>
        <toolbarItems>
            <barButtonItem style="plain" systemItem="flexibleSpace" id="XC5-Fu-Iwg"/>
        </toolbarItems>
        <navigationItem key="navigationItem" id="xNw-RF-rZk"/>
        <connections>
            <outlet property="dateTableView" destination="5Bb-Q9-0Xd" id="iBg-X7-Lmo"/>
            <segue destination="gjR-P9-utp" kind="push" identifier="userDidSelectAppointmentDate:"
                id="49k-h9-8hy"/>
        </connections>
    </viewController>
    <placeholder placeholderIdentifier="IBFirstResponder" id="6cc-GN-Nvg" userLabel="First Responder"
        sceneMemberID="firstResponder"/>
</objects>
<point key="canvasLocation" x="3493" y="1059"/>
</scene>
</scenes>
<resources>
    <image name="CameraButton.png" width="22" height="16"/>
</resources>
<simulatedMetricsContainer key="defaultSimulatedMetrics">
    <simulatedStatusBarMetrics key="statusBar"/>
    <simulatedOrientationMetrics key="orientation"/>
    <simulatedScreenMetrics key="destination" type="retina4"/>
</simulatedMetricsContainer>
<inferredMetricsTieBreakers>
    <segue reference="xxn-gz-vEJ"/>
    <segue reference="49k-h9-8hy"/>
</inferredMetricsTieBreakers>
</document>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0" toolsVersion="4510" systemVersion="12F45"
    targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES">
    <dependencies>
        <deployment defaultVersion="1536" identifier="iOS"/>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="3742"/>
    </dependencies>
    <objects>
        <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner"/>
        <placeholder placeholderIdentifier="IBFirstResponder" id="-2" customClass="UIResponder"/>
        <collectionViewCell opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="center" id="gL9-
            ra-h6v" customClass="SBCollectionViewLocationCell">
            <rect key="frame" x="0.0" y="0.0" width="130" height="130"/>
            <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMinY="YES"/>
            <view key="contentView" opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="center">
                <rect key="frame" x="0.0" y="0.0" width="130" height="130"/>
                <autoresizingMask key="autoresizingMask"/>
                <subviews>
                    <imageView userInteractionEnabled="NO" contentMode="scaleToFill" horizontalHuggingPriority="251"
                        verticalHuggingPriority="251" fixedFrame="YES"
                        translatesAutoresizingMaskIntoConstraints="NO" id="E2C-ER-3aP">
                        <rect key="frame" x="0.0" y="0.0" width="130" height="130"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                    </imageView>
                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
                        horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES" text="Label"
                        textAlignment="center" lineBreakMode="tailTruncation" numberOfLines="2" baselineAdjustment=
                        "alignBaselines" adjustsFontSizeToFit="NO" preferredMaxLayoutWidth="90"
                        translatesAutoresizingMaskIntoConstraints="NO" id="VlS-jj-fv8">
                        <rect key="frame" x="20" y="0.0" width="90" height="40"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <fontDescription key="fontDescription" type="boldSystem" pointSize="12"/>
                        <color key="textColor" white="1" alpha="1" colorSpace="calibratedWhite"/>
                        <nil key="highlightedColor"/>
                    </label>
                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
                        horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES" text="Label"
                        textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines"
                        adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="Og1-su-qfm">
                        <rect key="frame" x="0.0" y="109" width="130" height="21"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <fontDescription key="fontDescription" type="system" pointSize="9"/>
                        <color key="textColor" white="1" alpha="1" colorSpace="calibratedWhite"/>
                        <nil key="highlightedColor"/>
                    </label>
                </subviews>
                <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>
            </view>
            <connections>
                <outlet property="footnoteLabel" destination="Og1-su-qfm" id="axc-YG-d5m"/>
                <outlet property="imageView" destination="E2C-ER-3aP" id="tA6-ps-RMs"/>
                <outlet property="titleLabel" destination="VlS-jj-fv8" id="r7B-p1-VV6"/>
            </connections>
        </collectionViewCell>
    </objects>
</document>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0" toolsVersion="4510" systemVersion="13A603"
    targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES">
    <dependencies>
        <deployment defaultVersion="1536" identifier="iOS"/>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="3742"/>
    </dependencies>
    <objects>
        <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner"/>
        <placeholder placeholderIdentifier="IBFirstResponder" id="-2" customClass="UIResponder"/>
        <tableViewCell contentMode="scaleToFill" selectionStyle="default" indentationWidth="10" id="NYa-K5-q1I"
            customClass="SBHomeTableViewCell">
            <rect key="frame" x="0.0" y="0.0" width="320" height="50"/>
            <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
            <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES"
                contentMode="center" tableViewCell="NYa-K5-q1I" id="74k-Yh-Ub9">
                <rect key="frame" x="0.0" y="0.0" width="320" height="49"/>
                <autoresizingMask key="autoresizingMask"/>
                <subviews>
                    <imageView userInteractionEnabled="NO" contentMode="scaleToFill" horizontalHuggingPriority="251"
                        verticalHuggingPriority="251" translatesAutoresizingMaskIntoConstraints="NO" id="CZ8-hT-
                        MkT">
                        <rect key="frame" x="0.0" y="0.0" width="50" height="49"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <constraints>
                            <constraint firstAttribute="width" constant="50" id="26u-3c-fAp"/>
                        </constraints>
                    </imageView>
                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
                        horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Label" lineBreakMode=
                        "tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
                        translatesAutoresizingMaskIntoConstraints="NO" id="E7s-MS-TIh">
                        <rect key="frame" x="60" y="17" width="250" height="15"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <fontDescription key="fontDescription" name="HelveticaNeue-Bold" family="Helvetica Neue"
                            pointSize="15"/>
                        <nil key="highlightedColor"/>
                    </label>
                </subviews>
                <constraints>
                    <constraint firstItem="E7s-MS-TIh" firstAttribute="centerY" secondItem="CZ8-hT-MkT"
                        secondAttribute="centerY" id="C0d-6j-fGd"/>
                    <constraint firstItem="E7s-MS-TIh" firstAttribute="leading" secondItem="CZ8-hT-MkT"
                        secondAttribute="trailing" constant="10" id="Cj6-EW-AKN"/>
                    <constraint firstItem="E7s-MS-TIh" firstAttribute="top" secondItem="74k-Yh-Ub9"
                        secondAttribute="top" constant="17" id="K1h-xJ-Sak"/>
                    <constraint firstItem="CZ8-hT-MkT" firstAttribute="leading" secondItem="74k-Yh-Ub9"
                        secondAttribute="leading" id="QOE-Qi-fXe"/>
                    <constraint firstItem="CZ8-hT-MkT" firstAttribute="top" secondItem="74k-Yh-Ub9"
                        secondAttribute="top" id="cs9-ps-VgK"/>
                    <constraint firstAttribute="bottom" secondItem="CZ8-hT-MkT" secondAttribute="bottom" id="iNc-Nd-
                        xJT"/>
                    <constraint firstAttribute="trailing" secondItem="E7s-MS-TIh" secondAttribute="trailing"
                        constant="10" id="onM-ja-nq1"/>
                </constraints>
            </tableViewCellContentView>
            <connections>
                <outlet property="actionImageView" destination="CZ8-hT-MkT" id="gvu-1c-0Mg"/>
                <outlet property="actionTextLabel" destination="E7s-MS-TIh" id="T9v-Ae-05x"/>
            </connections>
        </tableViewCell>
    </objects>
</document>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0" toolsVersion="4510" systemVersion="12F37"
    targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES">
    <dependencies>
        <deployment defaultVersion="1536" identifier="iOS"/>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="3742"/>
    </dependencies>
    <objects>
        <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner"/>
        <placeholder placeholderIdentifier="IBFirstResponder" id="-2" customClass="UIResponder"/>
        <tableViewCell contentMode="scaleToFill" selectionStyle="default" accessoryType="disclosureIndicator"
            indentationWidth="10" id="iCQ-1Y-sPT" customClass="SBReservationOverviewCell">
            <rect key="frame" x="0.0" y="0.0" width="320" height="100"/>
            <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
            <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES"
                contentMode="center" tableViewCell="iCQ-1Y-sPT" id="dCR-Is-oPa">
                <rect key="frame" x="0.0" y="0.0" width="287" height="99"/>
                <autoresizingMask key="autoresizingMask"/>
                <subviews>
                    <imageView userInteractionEnabled="NO" contentMode="scaleToFill" horizontalHuggingPriority="251"
                        verticalHuggingPriority="251" fixedFrame="YES"
                        translatesAutoresizingMaskIntoConstraints="NO" id="WSJ-Cw-X7z">
                        <rect key="frame" x="9" y="10" width="80" height="80"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                    </imageView>
                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
                        horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES" text="Label"
                        lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
                        translatesAutoresizingMaskIntoConstraints="NO" id="zrn-LT-tVs">
                        <rect key="frame" x="97" y="10" width="200" height="21"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <fontDescription key="fontDescription" type="system" pointSize="17"/>
                        <nil key="highlightedColor"/>
                    </label>
                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
                        horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES" text="Label"
                        lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
                        translatesAutoresizingMaskIntoConstraints="NO" id="igZ-yc-dac">
                        <rect key="frame" x="97" y="29" width="200" height="21"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <fontDescription key="fontDescription" type="system" pointSize="17"/>
                        <nil key="highlightedColor"/>
                    </label>
                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
                        horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES" text="Label"
                        lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
                        translatesAutoresizingMaskIntoConstraints="NO" id="klo-G3-tQg">
                        <rect key="frame" x="97" y="69" width="200" height="21"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <fontDescription key="fontDescription" type="system" pointSize="17"/>
                        <nil key="highlightedColor"/>
                    </label>
                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
                        horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES" text="Label"
                        lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
                        translatesAutoresizingMaskIntoConstraints="NO" id="YeS-rB-lxA">
                        <rect key="frame" x="97" y="47" width="200" height="21"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <fontDescription key="fontDescription" type="system" pointSize="17"/>
                        <nil key="highlightedColor"/>
                    </label>
                </subviews>
            </tableViewCellContentView>
            <connections>
                <outlet property="reservationDateLabel" destination="igZ-yc-dac" id="oNs-tU-VRS"/>
                <outlet property="reservationImage" destination="WSJ-Cw-X7z" id="dYz-Nu-JKQ"/>
                <outlet property="reservationNameLabel" destination="zrn-LT-tVs" id="9Ov-eO-CCV"/>
                <outlet property="reservationStatusLabel" destination="klo-G3-tQg" id="2Lk-pA-vbj"/>
                <outlet property="reservationTimeLabel" destination="YeS-rB-lxA" id="1lF-d9-c4d"/>
            </connections>
        </tableViewCell>
    </objects>
</document>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0" toolsVersion="4510" systemVersion="13A603"
    targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES">
    <dependencies>
        <deployment defaultVersion="1536" identifier="iOS"/>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="3742"/>
    </dependencies>
    <objects>
        <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner"/>
        <placeholder placeholderIdentifier="IBFirstResponder" id="-2" customClass="UIResponder"/>
        <tableViewCell contentMode="scaleToFill" selectionStyle="default" indentationWidth="10" id="rRN-Hz-q15"
            customClass="SBReservationTableViewCell">
            <rect key="frame" x="0.0" y="0.0" width="320" height="50"/>
            <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
            <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES"
                contentMode="center" tableViewCell="rRN-Hz-q15" id="s0G-7V-Sgo">
                <rect key="frame" x="0.0" y="0.0" width="320" height="49"/>
                <autoresizingMask key="autoresizingMask"/>
                <subviews>
                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
                        horizontalHuggingPriority="251" text="Label" lineBreakMode="tailTruncation"
                        baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
                        translatesAutoresizingMaskIntoConstraints="NO" id="rg2-Ir-52i">
                        <rect key="frame" x="20" y="9" width="253" height="17"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <fontDescription key="fontDescription" type="boldSystem" pointSize="14"/>
                        <color key="textColor" cocoaTouchSystemColor="darkTextColor"/>
                        <nil key="highlightedColor"/>
                    </label>
                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
                        horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Label" lineBreakMode=
                        "tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
                        translatesAutoresizingMaskIntoConstraints="NO" id="TgF-CL-S0d">
                        <rect key="frame" x="20" y="20" width="253" height="27"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <fontDescription key="fontDescription" type="system" pointSize="11"/>
                        <color key="textColor" white="0.66666666666666663" alpha="1" colorSpace="calibratedWhite"/>
                        <nil key="highlightedColor"/>
                    </label>
                    <imageView userInteractionEnabled="NO" contentMode="scaleToFill" horizontalHuggingPriority="251"
                        verticalHuggingPriority="251" image="VerzoekOverzichtKruisje.png"
                        translatesAutoresizingMaskIntoConstraints="NO" id="AUL-G2-gfF">
                        <rect key="frame" x="299" y="11" width="11" height="11"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                    </imageView>
                    <activityIndicatorView hidden="YES" opaque="NO" contentMode="scaleToFill"
                        horizontalHuggingPriority="750" verticalHuggingPriority="750" hidesWhenStopped="YES" style=
                        "gray" translatesAutoresizingMaskIntoConstraints="NO" id="a5u-ZB-tnm">
                        <rect key="frame" x="280" y="15" width="20" height="20"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                    </activityIndicatorView>
                </subviews>
                <constraints>
                    <constraint firstItem="rg2-Ir-52i" firstAttribute="leading" secondItem="s0G-7V-Sgo"
                        secondAttribute="leading" constant="20" id="4aa-rF-6za"/>
                    <constraint firstAttribute="bottom" secondItem="TgF-CL-S0d" secondAttribute="bottom"
                        constant="2" id="7vV-DQ-8u0"/>
                    <constraint firstAttribute="bottom" relation="greaterThanOrEqual" secondItem="a5u-ZB-tnm"
                        secondAttribute="bottom" constant="14" id="Zea-ec-S46"/>
                    <constraint firstAttribute="trailing" secondItem="a5u-ZB-tnm" secondAttribute="trailing"
                        constant="20" id="b0Z-m9-EnZ"/>
                    <constraint firstAttribute="trailing" secondItem="TgF-CL-S0d" secondAttribute="trailing"
                        constant="47" id="e3s-Tk-A4z"/>
                    <constraint firstItem="TgF-CL-S0d" firstAttribute="top" secondItem="s0G-7V-Sgo"
                        secondAttribute="top" constant="20" symbolic="YES" id="hYs-Wm-P5b"/>
                    <constraint firstItem="rg2-Ir-52i" firstAttribute="trailing" secondItem="TgF-CL-S0d"
                        secondAttribute="trailing" id="j1B-GO-UcC"/>
                    <constraint firstItem="TgF-CL-S0d" firstAttribute="leading" secondItem="rg2-Ir-52i"
                        secondAttribute="leading" id="l8L-yh-5di"/>
                    <constraint firstItem="a5u-ZB-tnm" firstAttribute="top" relation="greaterThanOrEqual"
                        secondItem="s0G-7V-Sgo" secondAttribute="top" constant="15" id="lJv-zN-al0"/>
                    <constraint firstItem="AUL-G2-gfF" firstAttribute="top" secondItem="s0G-7V-Sgo"
                        secondAttribute="top" constant="11" id="m2R-3e-Swy"/>
                    <constraint firstItem="rg2-Ir-52i" firstAttribute="top" secondItem="s0G-7V-Sgo"
                        secondAttribute="top" constant="9" id="mri-we-XSu"/>
                    <constraint firstAttribute="right" secondItem="AUL-G2-gfF" secondAttribute="right" constant="10"
                        id="thF-Ib-Bo6"/>
                </constraints>
            </tableViewCellContentView>
            <connections>
                <outlet property="activityIndicator" destination="a5u-ZB-tnm" id="FDy-v0-PAp"/>
                <outlet property="crossImageView" destination="AUL-G2-gfF" id="f91-r4-A8Q"/>
                <outlet property="defectLabel" destination="rg2-Ir-52i" id="GBu-oS-FW8"/>
                <outlet property="timeLabel" destination="TgF-CL-S0d" id="TDG-3D-hQS"/>
            </connections>
        </tableViewCell>
    </objects>
```

```
    <resources>
        <image name="VerzoekOverzichtKruisje.png" width="11" height="11"/>
    </resources>
</document>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0" toolsVersion="4510" systemVersion="13A603"
    targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES">
    <dependencies>
        <deployment defaultVersion="1536" identifier="iOS"/>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="3742"/>
    </dependencies>
    <objects>
        <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner"/>
        <placeholder placeholderIdentifier="IBFirstResponder" id="-2" customClass="UIResponder"/>
        <tableViewCell contentMode="scaleToFill" selectionStyle="default" indentationWidth="10" id="nJ1-CF-rIX"
            customClass="SBSelectableOptionTableViewCell">
            <rect key="frame" x="0.0" y="0.0" width="320" height="44"/>
            <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
            <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES"
                contentMode="center" tableViewCell="nJ1-CF-rIX" id="cgH-GY-N08">
                <rect key="frame" x="0.0" y="0.0" width="320" height="43"/>
                <autoresizingMask key="autoresizingMask"/>
                <subviews>
                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
                        horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Label" lineBreakMode=
                        "tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
                        translatesAutoresizingMaskIntoConstraints="NO" id="MF7-SP-aXe">
                        <rect key="frame" x="20" y="11" width="290" height="21"/>
                        <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
                        <fontDescription key="fontDescription" name="HelveticaNeue-Bold" family="Helvetica Neue"
                            pointSize="13"/>
                        <color key="textColor" cocoaTouchSystemColor="darkTextColor"/>
                        <nil key="highlightedColor"/>
                    </label>
                </subviews>
                <constraints>
                    <constraint firstItem="MF7-SP-aXe" firstAttribute="top" secondItem="cgH-GY-N08"
                        secondAttribute="top" constant="11" id="Vxw-ms-cSX"/>
                    <constraint firstItem="MF7-SP-aXe" firstAttribute="centerY" secondItem="cgH-GY-N08"
                        secondAttribute="centerY" id="hrw-Bk-SH7"/>
                    <constraint firstItem="MF7-SP-aXe" firstAttribute="leading" secondItem="cgH-GY-N08"
                        secondAttribute="leading" constant="20" id="kCQ-wT-hbv"/>
                    <constraint firstAttribute="trailing" secondItem="MF7-SP-aXe" secondAttribute="trailing"
                        constant="10" id="m70-el-3Kf"/>
                </constraints>
            </tableViewCellContentView>
            <connections>
                <outlet property="optionLabel" destination="MF7-SP-aXe" id="ByD-Yt-mbu"/>
            </connections>
        </tableViewCell>
    </objects>
</document>
```

```php
<?php
/**
 * OAMobileAddMaintenanceAPIMethod
 *
 * Use basic authentication to authorize and receive a unique access token.
 *
 * default call:
 * /api/v1/addMaintenance/
 *
 */
class OAMobileAddMaintenanceAPIMethod extends Sef_Rest_Method_MethodAbstract
{
    /**
     * Retrieve an indicator if this method should respond to the given request or not.
     *
     * @param Sef_Rest_Request $request
     * @return bool
     */
    public function shouldRespondToRequest(Sef_Rest_Request $request)
    {
        // set the request content type
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_HTML);

        // build the request url for this method
        $requestUrl = sprintf('/^(%1$saddMaintenance(\/{0,1}))+$/', preg_quote($this->getService()->getBaseUrl(), '/'));

        // validate if the method type and request matches
        return $request->getMethodType() == Sef_Rest_Request::METHOD_POST &&
        preg_match($requestUrl, $request->getRequestUrl());
    }

    /**
     * Retrieve the Sef_Rest_Content_JsonSendContent object for this method.
     *
     * @param Sef_Rest_Request $request
     * @return Sef_Rest_Content_JsonSendContent
     */
    public function getContentByRequest(Sef_Rest_Request $request)
    {
        // create a new REST JSend return content object
        $content = new Sef_Rest_Content_JsonSendContent();

        // check for all required values to be uploaded
        if ($request->getPostParameter('defect_id', null) != null &&
            $request->getPostParameter('user_id', null) != null) {

            // retrieve the authenticated account, this is a OM_UuidAuthentication object
            $omUuidAuthentication = $this->getService()->getAuthenticatedAccount();

            // when the user does not have access to edit this maintenance user return false
            if (!$omUuidAuthentication->hasAccessToMaintenanceUserWithId($request->getPostParameter('user_id', null))) {

                // authentication failed, return false with authentication error
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                    ->setMessage('oa.maintenance.validation.access.denied');

                // return the created content
                $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
                return $content;
            }

            // try to fetch a user with the given ID
            try {
                // fetch the user
                $omMaintenanceUser = OM_MaintenanceUser::getById($request->getPostParameter('user_id', 0));
                $omDefect = OM_Defect::getById($request->getPostParameter('defect_id', 0));

                // set default values for op;tional values
                $vfsFileId = null;

                // check for an uploaded image, when uploaded and no error has occured continue to writing the file
                if (isset($_FILES['userfile'])) {
                    // when no error has occured store the file
                    if (is_uploaded_file($_FILES['userfile']['tmp_name']) && $_FILES['userfile']['error'] == UPLOAD_ERR_OK) {

                        // create the image name, namespace and company id
                        $fileName       = sprintf('maintenance-%1$s.jpg', date('d-m-Y H:i:s'));
                        $namespace      = 'public';
                        $companyId      = Sef_Core::getInstance()->getCompany()->getCompanyId();

                        // create a file system object with the provided namespace and company id
                        $vsfFileSystem  = new Filesystem($namespace, $companyId);

                        // default usage for folder is 2, store the image on disk
                        $vfsFolderId    = 2;
                        $vfsFileId      = $vsfFileSystem->storeFile($_FILES['userfile']['tmp_name'], $fileName, $vfsFolderId, $companyId);

                        // retreive the saved file object from the database
                        $vfsFile        = VFS_File::getById($vfsFileId);

                        // set the new filename and save to the database
                        $vfsFile->setName($fileName);
                        $vfsFile->save();
                    }
                    else
                    {
                        // an error occured while uploading the image
                        $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                            ->setMessage('oa.maintenance.image.failed_upload');
                        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
                        return $content;
                    }
                }

                // validate whether the description is not larger than the allowed length
                $description = $request->getPostParameter('description', '');
                if (strlen($description) > 256) {
                    // an error occured while uploading the image
                    $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                        ->setMessage('oa.maintenance.validation.description.invalid');
                    $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
                    return $content;
                }

                // create the new maintenance and configure the values
                $newOmMaintenance = new OM_Maintenance();
                $newOmMaintenance->setOmDefect($omDefect);
                $newOmMaintenance->setDescription($description);
                $newOmMaintenance->setOmMaintenanceUser($omMaintenanceUser);
```

```php
                // add the image file id when set
                if ($vfsFileId !== null) {
                    $newOmMaintenance->setVfsFileId($vfsFileId);
                }

                try {
                    // insert the maintenance user object
                    $newOmMaintenance->insert();

                    // when the final maintenance has been uploaded send a notification
                    if (strcmp($request->getPostParameter('is_final_maintenance', 'false'), 'true') == 0) {

                        // check for the apns device to be available for this user, when it is a notification can be sent
                        if ($omMaintenanceUser->getApnsDevice() !== null) {
                            // send the user a notification to let know that all maintenances have been received and are in a pending state
                            $apnsNotification = new APNS_DeviceNotification();
                            $apnsNotification->setText("oa.maintenance.notification.all_received");
                            $apnsNotification->setNotificationDatetime(null);
                            $apnsNotification->setApnsDevice($omMaintenanceUser->getApnsDevice());
                            $apnsNotification->setStatus("queued");

                            // insert the notification which will be sent when possible
                            $apnsNotification->insert();
                        }

                        // send the user a new message with information about the successfull received maintenance(s)
                        $uploadedMessage = new OM_MaintenanceUserSuccessfullyUploadedMessage($omMaintenanceUser);
                        $uploadedMessage->send();

                        // insert went successfully, return success status
                        $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_SUCCESS)
                            ->setCode(Sef_Rest_Response::STATUS_CODE_OK);
                    }
                }
                catch (Exception $exception) {
                    // the maintenance could not be uploaded, return error
                    $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                        ->setMessage($exception->getMessage());
                }
            }
            catch (Exception $exception)
            {
                // the user does not exist, exit
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                    ->setMessage($exception->getMessage());
            }
        }
        else
        {
            // not all required values have been set, show error
            $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                ->setMessage('oa.maintenance.validation.missing_values');
        }

        // return the created content
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
        return $content;
    }

    /**
     * Retrieve an indicator if the method requires authentication before it may be executed
     *
     * @return bool
     */
    public function doesRequireAuthentication()
    {
        return true;
    }
}
```

```php
<?php
/**
 * OAMobileAddressLookUpAPIMethod
 *
 * Use basic authentication to authorize and receive a unique access token.
 *
 * default call:
 * /api/v1/lookUpAddress/
 *
 */
class OAMobileAddressLookUpAPIMethod extends Sef_Rest_Method_MethodAbstract
{
    /**
     * Retrieve an indicator if this method should response to the given request or not.
     *
     * @param Sef_Rest_Request $request
     * @return bool
     */
    public function shouldRespondToRequest(Sef_Rest_Request $request)
    {
        // set the request type to html
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_HTML);

        // build the request url for this method
        $requestUrl = sprintf('/^(%slookUpAddress(\/{0,1}))+$/', preg_quote($this->getService()->getBaseUrl(), '/'));

        // validate if the method type and request matches
        return $request->getMethodType() == Sef_Rest_Request::METHOD_POST &&
        preg_match($requestUrl, $request->getRequestUrl());
    }

    /**
     * Retrieve the Sef_Rest_Content_JsonSendContent object for this method.
     *
     * @param Sef_Rest_Request $request
     * @return Sef_Rest_Content_JsonSendContent
     */
    public function getContentByRequest(Sef_Rest_Request $request)
    {
        // create a new REST JSend return content object
        $content = new Sef_Rest_Content_JsonSendContent();

        // check wether the required post values are set
        if ($request->getPostParameter('zipcode', null) !== null &&
            $request->getPostParameter('house_number', null) !== null) {

            // escape strings
            $zipcode = $request->getPostParameter('zipcode');
            $houseNumber = intval($request->getPostParameter('house_number'));
            $addition = $request->getPostParameter('addition', null);

            // validate whether the zipcode is valid format
            if (!Sef_Helpers_Validator::isValidDutchZipcode($zipcode)|| !($houseNumber >= 1 && $houseNumber <= 20000)) {
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                    ->setMessage('oa.address.validation.invalid_data');
                $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
                return $content;
            }

            // create address lookup object to validate the given info
            $address = new OM_AddressLookUp();
            $validatedLocation = $address->validateAddress(strtoupper(str_replace(' ', '', $zipcode)), $houseNumber, $addition);

            // when the validated location is found, no errors occured
            if ($validatedLocation !== null) {
                // a correct location is found
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_SUCCESS)
                    ->setCode(Sef_Rest_Response::STATUS_CODE_OK)
                    ->setData(array(
                        'zipcode' => $validatedLocation->getZipcode(),
                        'house_number' => $validatedLocation->getHouseNumber(),
                        'house_number_addition' => $validatedLocation->getHouseNumberAddition(),
                        'street' => $validatedLocation->getStreet(),
                        'city' => $validatedLocation->getCity(),
                        'latitude' => $validatedLocation->getLatitude(),
                        'longitude' => $validatedLocation->getLongitude())
                            );
            }
            else
            {
                // get the error object
                $error = $address->getValidationError();

                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                    ->setMessage($error[OM_AddressLookUp::APIAddressErrorDescription]);
            }
        }
        else
        {
            // return response with error that not all values have been set
            $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                ->setMessage('oa.address.validation.missing_values');
        }

        // return the created content
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
        return $content;
    }

    /**
     * Retrieve an indicator if the method requires authentication before it may be executed
     *
     * @return bool
     */
    public function doesRequireAuthentication()
    {
        return true;
    }
}
```

```php
<?php
/**
 * OAMobileAddUserAPIMethod
 *
 * Use basic authentication to authorize and receive a unique access token.
 *
 * default call:
 * /api/v1/addUser/
 *
 */
class OAMobileAddUserAPIMethod extends Sef_Rest_Method_MethodAbstract
{
    /**
     * Retrieve an indicator if this method should response to the given request or not.
     *
     * @param Sef_Rest_Request $request
     * @return bool
     */
    public function shouldRespondToRequest(Sef_Rest_Request $request)
    {
        // set the request type to html
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_HTML);

        // build the request url for this method
        $requestUrl = sprintf('/^(%saddUser(\/{0,1}))+$/', preg_quote($this->getService()->getBaseUrl(), '/'));

        // validate if the method type and request matches
        return $request->getMethodType() == Sef_Rest_Request::METHOD_POST &&
        preg_match($requestUrl, $request->getRequestUrl());
    }

    /**
     * Retrieve the Sef_Rest_Content_JsonSendContent object for this method.
     *
     * @param Sef_Rest_Request $request
     * @return Sef_Rest_Content_JsonSendContent
     */
    public function getContentByRequest(Sef_Rest_Request $request)
    {
        // create a new REST JSend return content object
        $content = new Sef_Rest_Content_JsonSendContent();

        // check wether all required values have been uploaded
        if ($request->getPostParameter('name') != null &&
            $request->getPostParameter('email') != null &&
            $request->getPostParameter('phone_number') != null &&
            $request->getPostParameter('zipcode') != null &&
            $request->getPostParameter('house_number', 0) != 0 &&
            $request->getPostParameter('customer_id') != null &&
            $request->getPostParameter('date', 0) != 0 &&
            $request->getPostParameter('daypart_id') != null) {

            try {
                // start validating the data, when not valid a exception is thrown
                $this->validateRequiredValues($request);
            }
            catch (Exception $exception) {
                // set the message to be the exception message
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                    ->setMessage($exception->getMessage());

                // set content type and return the data
                $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
                return $content;
            }

            try {
                // get the customer
                $rmCustomer = RM_Customer::getById($request->getPostParameter('customer_id', 0));
                if (!$rmCustomer->getIsSelectableInMobileApplication()) {
                    throw new Exception('Customer not available for mobile application.');
                }
            }
            catch (Exception $exception)
            {
                // when the customer could not be found return error
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                    ->setMessage("oa.maintenance.customer.invalid");
                $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
                return $content;

            }

            // retreive the availability object for the day and daypart
            $omAvailability = $this->getAvailabilityObjectForDateAndDaypart($request->getPostParameter('date'), $request-
>getPostParameter('daypart_id'));

            // there are still dayparts available, continue the request
            if ($omAvailability !== false) {

                // create a new maintenance user
                $newMaintenanceUser = new OM_MaintenanceUser();
                $newMaintenanceUser->setOmAvailability($omAvailability);

                // validate the address
                $addressValidator = new OM_AddressLookUp();
                $houseNumber = intval($request->getPostParameter('house_number'));
                $location =  $addressValidator->validateAddress($request->getPostParameter('zipcode'), $houseNumber, $request-
>getPostParameter('number_addition'));

                if ($location !== null) {
                    // configure the new user
                    $newMaintenanceUser->setName($request->getPostParameter('name'));
                    $newMaintenanceUser->setEmail($request->getPostParameter('email'));
                    $newMaintenanceUser->setPhoneNumber_1($request->getPostParameter('phone_number'));
                    $newMaintenanceUser->setPhoneNumber_2($request->getPostParameter('phone_number_optional'));
                    $newMaintenanceUser->setZipcode($location->getZipcode());
                    $newMaintenanceUser->setHouseNumber($location->getHouseNumber());
                    $newMaintenanceUser->setRmCustomer($rmCustomer);
                    $newMaintenanceUser->setAdressAddition($location->getHouseNumberAddition());
                    $newMaintenanceUser->setAdress($location->getStreet());
                    $newMaintenanceUser->setCity($location->getCity());
                    $newMaintenanceUser->setOmMaintenanceStateId(1);
                    $newMaintenanceUser->setIsProcessedInMer(false);
                    $newMaintenanceUser->setIsArchived(false);
                    $newMaintenanceUser->setOmUuidAuthentication($this->getService()->getAuthenticatedAccount());

                    // when the notification token is set create a notification object
                    if ($request->getPostParameter('notification_token') != null && strlen($request->getPostParameter('notification_token', '')) ==
64) {
```

```
                        $apnsDevice = new APNS_Device();
                        $apnsDevice->setDeviceToken($request->getPostParameter('notification_token'));
                        $apnsDevice->setRegistrationDatetime(Sef_Helpers_DateTime::getSqlDateByTimestamp(time()));

                        // insert the notification object and set the apns device reference in the maintenance user
                        $apnsDevice->insert();
                        $newMaintenanceUser->setApnsDevice($apnsDevice);
                    }

                    // try to insert the user
                    $userId = $newMaintenanceUser->insert();

                    // when successfully inserted the user continue
                    if ($userId > 0) {

                        // set the return content of this API method,
                        $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_SUCCESS)
                            ->setCode(Sef_Rest_Response::STATUS_CODE_OK)
                            ->setData(array('user_id' => $userId));
                    }
                    else
                    {
                        // the user could not be added, return error
                        $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                            ->setMessage('oa.maintenance_user.insert.failed');
                    }
                }
                else
                {
                    // the provided address is not valid
                    $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                        ->setMessage('oa.maintenance_user.address.invalid');
                }
            }
            else
            {
                // the date and daypart are not valid
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                    ->setMessage('oa.maintenance_user.date.invalid');
            }
        }
        else
        {
            // set the return content of this API method, not all values are provided
            $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                ->setMessage('oa.maintenance_user.missing.values');
        }

        // return the created content
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
        return $content;
    }

    /*
     * Retreives an available object for reservation, when no object could be fetched returns false, the reservation cannot be made
     * @param   int (unix timestamp)
     * @param   int (id of the selected daypart)
     * @result  OM_Availability | false
     */
    public function getAvailabilityObjectForDateAndDaypart($date, $daypartId)
    {
        // retrieve all available dates for maintenance, the availability count must be greater than 0 and the dates must be between the calculated
dates
        $possibleOMAvailabilities = OM_Availability::getAllWithSortByWhere(null,
            array(
                OM_Availability::DATE           => Sef_Helpers_DateTime::getSqlDateByTimestamp($date),
                OM_Availability::OM_DAY_PART_ID => $daypartId)
        );

        // when possible dates are found continue
        if (count($possibleOMAvailabilities) > 0) {

            // check for each possible date wether the availability count is lower than the amount of reservation
            foreach ($possibleOMAvailabilities as $possibleOMAvailability) {
                // get the counter for the current availability
                $currentCount = OM_MaintenanceUser::getCountAllByWhere(
                    array(
                    OM_MaintenanceUser::OM_AVAILABILITY_ID => $possibleOMAvailability->getOmAvailabilityId()
                ));

                // when there are fewer reservations return this object
                if ($currentCount < $possibleOMAvailability->getAvailabilityCount()) {
                    return $possibleOMAvailability;
                }
            }

            // when no object is available return false, not date is available for the given parameters
            return false;
        }
        else {
            return false;
        }
    }

    /**
     * Method validates all required and optional values which are no foreign keys to tables
     *
     * @param Sef_Rest_Request $request
     * @throws Exception
     */
    private function validateRequiredValues(Sef_Rest_Request $request)
    {
        // set the optional values
        $houseNumber = intval($request->getPostParameter('house_number', 0));
        $nameLength = strlen($request->getPostParameter('name', ''));

        // validate all required values
        if (!Sef_Helpers_Validator::isValidDutchZipcode($request->getPostParameter('zipcode'))) {
            throw new Exception('oa.maintenance.user.validation.zipcode.invalid');
        }

        // validate the house number to be between 1 and 20000
        if ($houseNumber < 1 && $houseNumber > 20000) {
            throw new Exception('oa.maintenance.user.validation.housenumber.invalid');
        }

        // validate the email adress
        if (!Sef_Helpers_Validator::isValidEmail($request->getPostParameter('email'))) {
            throw new Exception('oa.maintenance.user.validation.email.invalid');
        }
```

```
        // name must be between 1 and 128 characters
        if ($nameLength > 128 || $nameLength == 0) {
            throw new Exception('oa.maintenance.user.validation.name.invalid');
        }

        // validate the required phone number
        if (!Sef_Helpers_Validator::isValidDutchPhoneNumber($request->getPostParameter('phone_number', null))) {
            throw new Exception('oa.maintenance.user.phonenumber.invalid');
        }

        // validate the optional phone number when it is set
        if ($request->getPostParameter('phone_number_optional', null) !== null &&
            !Sef_Helpers_Validator::isValidDutchPhoneNumber($request->getPostParameter('phone_number_optional', null))) {
            throw new Exception('oa.maintenance.user.validation.phonenumber.optional.invalid');
        }
    }

    /**
     * Retrieve an indicator if the method requires authentication before it may be executed
     *
     * @return bool
     */
    public function doesRequireAuthentication()
    {
        return true;
    }
}
```

```php
<?php
/**
 * OAMobileAllCustomersAPIMethod
 *
 * Use basic authentication to authorize and receive a unique access token.
 *
 * default call:
 * /api/v1/allCustomers/
 *
 */
class OAMobileAllCustomersAPIMethod extends Sef_Rest_Method_MethodAbstract
{
    /**
     * Retrieve an indicator if this method should response to the given request or not.
     *
     * @param Sef_Rest_Request $request
     * @return bool
     */
    public function shouldRespondToRequest(Sef_Rest_Request $request)
    {
        // set JSON contentn type
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);

        // build the request url for this method
        $requestUrl = sprintf('/^(%1$sallCustomers(\/{0,1}))+$/', preg_quote($this->getService()->getBaseUrl(), '/'));

        // validate if the method type and request matches
        return $request->getMethodType() == Sef_Rest_Request::METHOD_GET &&
        preg_match($requestUrl, $request->getRequestUrl());
    }

    /**
     * Retrieve the Sef_Rest_Content_JsonSendContent object for this method.
     *
     * @param Sef_Rest_Request $request
     * @return Sef_Rest_Content_JsonSendContent
     */
    public function getContentByRequest(Sef_Rest_Request $request)
    {
        // create a new REST JSend return content object
        $content = new Sef_Rest_Content_JsonSendContent();

        // retrieve all available customers from the database where the is selectable in mobile application flag is true
        $jsonCustomers = array();
        $selectableRmCustomers = RM_Customer::getAllWithSortByWhere(null,
            array(
                RM_Customer::IS_SELECTABLE_IN_MOBILE_APPLICATION => true,
            )
        );
        // add each customer to the data array
        foreach ($selectableRmCustomers as $currentRmCustomer) {

            // each customer object consists of an id and description
            $jsonCustomers[] =
                array(
                'id'            => $currentRmCustomer->getRmCustomerId(),
                'description'   => $currentRmCustomer->getName()
            );
        }

        // set the return content of this API method
        $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_SUCCESS)
            ->setCode(Sef_Rest_Response::STATUS_CODE_OK)
            ->setData($jsonCustomers);

        // return the created content
        return $content;
    }

    /**
     * Retrieve an indicator if the method requires authentication before it may be executed
     *
     * @return bool
     */
    public function doesRequireAuthentication()
    {
        return true;
    }
}
```

```php
<?php
/**
 * OAMobileAuthenticationRequestAPIMethod.php
 *
 * @copyright   Copyright (c) 2007-2013 Solware B.V. (http://www.solware.nl)
 */
class OAMobileAuthenticationRequestAPIMethod extends Sef_Rest_Method_MethodAbstract
{
    /**
     * Retrieve an indicator if this method should response to the given request or not.
     *
     * @param Sef_Rest_Request $request
     * @return bool
     */
    public function shouldRespondToRequest(Sef_Rest_Request $request)
    {
        // set the request type to html
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_HTML);

        // build the request url for this method
        $requestUrl = sprintf('/^(%sauthenticationRequest(\/{0,1}))+$/', preg_quote($this->getService()->getBaseUrl(), '/'));

        // validate if the method type and request matches
        return $request->getMethodType() == Sef_Rest_Request::METHOD_POST &&
        preg_match($requestUrl, $request->getRequestUrl());
    }

    /**
     * Retrieve the Sef_Rest_Content_JsonSendContent object for this method.
     *
     * @param Sef_Rest_Request $request
     * @return Sef_Rest_Content_JsonSendContent
     */
    public function getContentByRequest(Sef_Rest_Request $request)
    {
        // create a new REST JSend return content object
        $content = new Sef_Rest_Content_JsonSendContent();

        // get the UUID to per created a authentication user for
        $uuid = $request->getPostParameter('UUID', null);

        // validate whether the UUID is set in the POST parameters and the length is exactly 36 characters
        if ($uuid !== null && strlen($uuid) === 36) {

            // load the already existing user
            $uuidAuthenticationUser = OM_UuidAuthentication::getFirstByUuidPassword($uuid, false);

            // when no accounts have been found create a new authentication account, else send the already existing data
            if ($uuidAuthenticationUser === null) {

                // create the new uuid account credentials
                $newUsername = Sef_Helpers_String::generateRandomString(64);
                $newToken = Sef_Helpers_String::generateRandomString(64);

                // create the actual account object
                $newUuidAuthentication = new OM_UuidAuthentication();
                $newUuidAuthentication->setUuidPassword($uuid);
                $newUuidAuthentication->setUuidUsername($newUsername);
                $newUuidAuthentication->setAccountIsBlocked(false);
                $newUuidAuthentication->setUuidLoginToken($newToken);

                // insert the new account
                if ($newUuidAuthentication->insert()) {

                    // provide the username and token in the data so the application can use these credentials for connecting
                    $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_SUCCESS)
                        ->setData(array(
                            'user'      =>      $newUsername,
                            'token'     =>      $newToken,
                        ));
                } else {

                    // provide the status code that token and username could not be provided
                    $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                        ->setMessage(__('om.uuid.validation.create.failed'))
                        ->setCode(1);
                }

            } else {

                // send the already existing content
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_SUCCESS)
                     ->setData(array(
                        'user'  =>  $uuidAuthenticationUser->getUuidUsername(),
                        'token' =>  $uuidAuthenticationUser->getUuidLoginToken(),
                ));
            }
        } else {

            // create status failed for invalid data
            $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_FAIL)
                ->setData(array(__('om.uuid.validation.invalid.uuid')));
        }

        // return the created content
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
        return $content;
    }

    /**
     * Retrieve an indicator if the method requires authentication before it may be executed
     *
     * @return bool
     */
    public function doesRequireAuthentication()
    {
        return false;
    }
}
```

```php
<?php
/**
 * OAMobileAvailableDatesAPIMethod
 *
 * Use basic authentication to authorize and receive a unique access token.
 *
 * default call:
 * /api/v1/getAvailableDates/
 *
 */
class OAMobileAvailableDatesAPIMethod extends Sef_Rest_Method_MethodAbstract
{
    // constants to indicated the amount of days from and to to load for being available
    const OAMaxFutureDays    =   31;
    const OAMinFutureDays    =   3;

    /**
     * Retrieve an indicator if this method should response to the given request or not.
     *
     * @param Sef_Rest_Request $request
     * @return bool
     */
    public function shouldRespondToRequest(Sef_Rest_Request $request)
    {
        // build the request url for this method
        $requestUrl = sprintf('/^(%1$sgetAvailableDates(\/{0,1}))+$/', preg_quote($this->getService()->getBaseUrl(), '/'));

        // validate if the method type and request matches
        return $request->getMethodType() == Sef_Rest_Request::METHOD_GET &&
        preg_match($requestUrl, $request->getRequestUrl());
    }

    /**
     * Retrieve the Sef_Rest_Content_JsonSendContent object for this method.
     *
     * @param Sef_Rest_Request $request
     * @return Sef_Rest_Content_JsonSendContent
     */
    public function getContentByRequest(Sef_Rest_Request $request)
    {
        // create a new REST JSend return content object
        $content = new Sef_Rest_Content_JsonSendContent();

        // retreive all available dayparts and process them to an array
        $dayPartArray = array();

        // loop through the day parst to process them to an associative array
        foreach (OM_DayPart::getAll() as $currentDayPart)
        {
            // the day pary can be associated to the ID key
            $dayPartArray[$currentDayPart->getOmDayPartId()] = array(
                'id'            => $currentDayPart->getOmDayPartId(),
                'description'   => $currentDayPart->getTimeDescription(),
                'start'         => $currentDayPart->getStartTime(),
                'end'           => $currentDayPart->getEndTime()
            );
        }

        // retreive all available dates for maintenance, the availability count must be greater than 0 and the dates must be between the calculated
dates
        $omAvailability = $this->getAvailabilityObjects();
        $currentDate = 0;
        $currentIndex = 0;
        $dataArray = array();

        // check for the availability array to be filled with possible dates
        if (is_array($omAvailability) && count($omAvailability) > 0) {

            // loop through all available dates
            for ($i = 0; $i < count($omAvailability); $i++) {
                // get the new date
                $newDate = Sef_Helpers_DateTime::getTimestampBySqlDate($omAvailability[$i]->getDate());

                // get the current daypart and add the available count
                $currentDaypart = $dayPartArray[$omAvailability[$i]->getOmDayPartId()];
                $currentDaypart['available'] = $omAvailability[$i]->getAvailabilityCount();

                // when the current date is not equal to the next object a new date object should be created
                if ($currentDate != $newDate) {
                    $dataArray[$currentIndex] = array('date' => $newDate, 'day_part' => array());
                    $dataArray[$currentIndex]['day_part'][] = $currentDaypart;
                    $currentIndex++;
                }
                else
                {
                    // append the current index of the array
                    $dataArray[$currentIndex - 1]['day_part'][] = $currentDaypart;
                }

                // set the new current date
                $currentDate = $newDate;
            }
        }

        // when no dates could be selected return error
        if (count($dataArray) == 0) {
            $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                ->setMessage('api.rest.available_dates.none')
                ->setCode(Sef_Rest_Response::STATUS_CODE_OK);
        }
        else
        {
            // set the return content of this API method,
            $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_SUCCESS)
                ->setCode(Sef_Rest_Response::STATUS_CODE_OK)
                ->setData($dataArray);
        }
        // return the created content
        return $content;
    }

    /*
     * Retreives an available object for reservation, when no object could be fetched returns false, the reservation cannot be made
     * @param   date (unix timestamp)
     * @param   daypartId (id of the selected daypart)
     * @result OM_Availability | false
    */
    private function getAvailabilityObjects()
    {
        // calculate the dates to search the database for, maintenance can be scheduled between the calculated dates
```

```
        $minimumDate = time() + (self::OAMinFutureDays * 86400);
        $maximumDate = $minimumDate + (self::OAMaxFutureDays * 86400);

        // retreive all available dates for maintenance, the availability count must be greater than 0 and the dates must be between the calculated
dates
        $omAvailability = OM_Availability::getAllWithSortByWhere(
            array(
                OM_Availability::DATE => DataObject::ASCENDING,
                OM_AvailabilityBase::OM_DAY_PART_ID.'.'.OM_DayPartBase::OM_DAY_PART_ID => DataObject::ASCENDING
            ),
            array(
                '[>]' . OM_Availability::AVAILABILITY_COUNT => 0,
                '[>=]' . OM_Availability::DATE => Sef_Helpers_DateTime::getSqlDateByTimestamp($minimumDate),
                '[<=]' . OM_Availability::DATE => Sef_Helpers_DateTime::getSqlDateByTimestamp($maximumDate)
            ));


        // when possible dates are found continue
        if (count($omAvailability) > 0) {

            // create the container for all available objects
            $validatedAvailableObjects = array();

            // check for each possible date wether the availability count is lower than the amount of reservation
            for ($i = 0; $i < count($omAvailability); $i++)
            {
                // get the counter for the current availability
                $currentCount = OM_MaintenanceUser::getCountAllByWhere(array(
                    OM_MaintenanceUser::OM_AVAILABILITY_ID => $omAvailability[$i]->getOmAvailabilityId()
                ));

                // when there are fewer reservations return this object
                if ($currentCount < $omAvailability[$i]->getAvailabilityCount()) {

                    // update the availability count to show the correct amount of users that have reserved
                    $omAvailability[$i]->setAvailabilityCount($omAvailability[$i]->getAvailabilityCount() - $currentCount);
                    $validatedAvailableObjects[] = $omAvailability[$i];
                }
            }

            // when objects are in the array return the array, else return false
            if (count($validatedAvailableObjects) > 0) {
                return $validatedAvailableObjects;
            }
            else {
                return false;
            }
        }
        else {
            return false;
        }
    }

    /**
     * Retrieve an indicator if the method requires authentication before it may be executed
     *
     * @return bool
     */
    public function doesRequireAuthentication()
    {
        return true;
    }
}
```

```php
<?php
/**
 * OAMobileCancelReservationAPIMethod
 *
 * Use basic authentication to authorize and receive a unique access token.
 *
 * default call:
 * /api/v1/cancelReservation/
 *
 */

class OAMobileCancelReservationAPIMethod extends Sef_Rest_Method_MethodAbstract
{
    /**
     * Retrieve an indicator if this method should response to the given request or not.
     *
     * @param Sef_Rest_Request $request
     * @return bool
     */
    public function shouldRespondToRequest(Sef_Rest_Request $request)
    {
        // set the request type to html
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_HTML);

        // build the request url for this method
        $requestUrl = sprintf('/^(%1$scancelReservation(\/{0,1}))+$/', preg_quote($this->getService()->getBaseUrl(), '/'));

        // validate if the method type and request matches
        return $request->getMethodType() == Sef_Rest_Request::METHOD_POST &&
        preg_match($requestUrl, $request->getRequestUrl());
    }

    /**
     * Retrieve the Sef_Rest_Content_JsonSendContent object for this method.
     *
     * @param Sef_Rest_Request $request
     * @return Sef_Rest_Content_JsonSendContent
     */
    public function getContentByRequest(Sef_Rest_Request $request)
    {
        // create a new REST JSend return content object
        $content = new Sef_Rest_Content_JsonSendContent();

        // get the reservation id
        $omMaintenanceUserId = $request->getPostParameter('reservation_id', null);

        if ($omMaintenanceUserId !== null) {

            // get the current authenticated account
            $omUuidAuthentication = $this->getService()->getAuthenticatedAccount();

            // when the user does not have access to cancel this maintenance stop executing
            if (!$omUuidAuthentication->hasAccessToMaintenanceUserWithId($request->getPostParameter('reservation_id', null))) {

                // authentication failed, return false with authentication error
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                    ->setMessage('oa.maintenance.validation.access.denied');

                // return the created content
                $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
                return $content;
            }

            try {
                // try to retrieve the maintenance user by the given ID
                $omMaintenanceUser = OM_MaintenanceUser::getById($omMaintenanceUserId);

                // set cancelled state and set archived
                $omMaintenanceUser->setOmMaintenanceStateId(OM_MaintenanceState::CANCELLED_STATE);
                $omMaintenanceUser->setIsArchived(true);

                // reset the availability so the date this maintenance was scheduled on will be free again for other users
                $omMaintenanceUser->setOmAvailability(null);
                $omMaintenanceUser->save();

                // create and send the message
                $omCancelledMessage = new OM_MaintenanceUserCancelledMessage($omMaintenanceUser);
                $omCancelledMessage->send();

                // create a history state object
                $omStateHistory = new OM_MaintenanceUserStateHistory();
                $omStateHistory->setOmMaintenanceStateId(OM_MaintenanceState::CANCELLED_STATE);
                $omStateHistory->setOmMaintenanceUser($omMaintenanceUser);
                $omStateHistory->setStateCorrespondingEmailWasSent(true);
                $omStateHistory->setStateChangedDate(Sef_Helpers_DateTime::getSqlDateTimeByTimestamp(time()));

                // save the history state object
                $omStateHistory->save();

                // set success code
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_SUCCESS)
                    ->setCode(Sef_Rest_Response::STATUS_CODE_OK)
                    ->setData(array('id' => $omMaintenanceUser->getOmMaintenanceUserId()));
            } catch (LoadException $exception) {
                // set the message to id not found
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                    ->setMessage('oa.reservation.cancel.invalid.id');
            }
        } else {
            $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                ->setMessage('oa.reservation.cancel.missing.id');
        }

        // set the type of content to return and return the content
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
        return $content;

    }

    /**
     * Retrieve an indicator if the method requires authentication before it may be executed
     *
     * @return bool
     */
    public function doesRequireAuthentication()
    {
        return true;
    }
}
```

```php
<?php
/**
 * OAMobileCheckstateAPIMethod
 *
 * Use basic authentication to authorize and receive a unique access token.
 *
 * default call:
 * /api/v1/checkState/
 *
 */

class OAMobileCheckStateAPIMethod extends Sef_Rest_Method_MethodAbstract
{
    /**
     * Retrieve an indicator if this method should response to the given request or not.
     *
     * @param Sef_Rest_Request $request
     * @return bool
     */
    public function shouldRespondToRequest(Sef_Rest_Request $request)
    {
        // set the request type to html
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_HTML);

        // build the request url for this method
        $requestUrl = sprintf('/^(%1$scheckState(\/{0,1}))+$/', preg_quote($this->getService()->getBaseUrl(), '/'));

        // validate if the method type and request matches
        return $request->getMethodType() == Sef_Rest_Request::METHOD_POST &&
        preg_match($requestUrl, $request->getRequestUrl());
    }

    /**
     * Retrieve the Sef_Rest_Content_JsonSendContent object for this method.
     *
     * @param Sef_Rest_Request $request
     * @return Sef_Rest_Content_JsonSendContent
     */
    public function getContentByRequest(Sef_Rest_Request $request)
    {
        // get all reservation ids that are provided via the post parameters
        $reservationIdString = $request->getPostParameter('reservation_list', null);

        // create a new REST JSend return content object
        $content = new Sef_Rest_Content_JsonSendContent();

        // when the list is provided continue
        if ($reservationIdString !== null) {

            // remove all ',' separators to retrieve all ids in the array
            $reservationIds = explode(',', $reservationIdString);
            $resultArray = array();

            // get the current authenticated account
            $omUuidAuthentication = $this->getService()->getAuthenticatedAccount();

            foreach ($reservationIds as $reservationId) {

                // when the user does not have access to read the state of this reservation skip this one by not adding the result data
                if (!$omUuidAuthentication->hasAccessToMaintenanceUserWithId($reservationId)) {

                    // continue to the next maintenance state check
                    continue;
                }

                // get the manitenance user for the given reservarion id ()
                try {
                    // cast the reservation id
                    $reservationId = (int)$reservationId;

                    // get the maintenance user for the provided id
                    $currentOmMaintenanceUser = OM_MaintenanceUser::getById(intval($reservationId));
                    $currentState = array();

                    switch ($currentOmMaintenanceUser->getOmMaintenanceStateId()) {
                        case OM_MaintenanceState::RESERVED_STATE:
                            $currentState = array(
                                'id'    =>  $reservationId,
                                'state' =>  'reserved',
                            );
                            break;

                        case OM_MaintenanceState::CONFIRMED_STATE:
                            $currentState = array(
                                'id'    =>  $reservationId,
                                'state' =>  'approved',
                            );
                            break;

                        case OM_MaintenanceState::DENIED_STATE:
                            $currentState = array(
                                'id'    =>  $reservationId,
                                'state' =>  'denied',
                            );
                            break;

                        case OM_MaintenanceState::FINISHED_STATE:
                            $currentState = array(
                                'id'    =>  $reservationId,
                                'state' =>  'finished',
                            );
                            break;

                        case OM_MaintenanceState::CANCELLED_STATE:
                            $currentState = array(
                                'id'    =>  $reservationId,
                                'state' =>  'cancelled',
                            );
                            break;

                        default:
                            $currentState = array(
                                'id'    =>  $reservationId,
                                'state' =>  'unknown',
                            );
                            break;
                    }

                    // add the current state to the array
                    $resultArray[] = $currentState;
```

```
                } catch (LoadException $exception) {
                    // add error to the array
                    $resultArray[] = array(
                        'id'    =>  $reservationId,
                        'state' =>  'unknown',
                    );
                }
            }

            // create the content to return
            $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_SUCCESS)
                ->setData($resultArray)
                ->setCode(Sef_Rest_Response::STATUS_CODE_OK);
        } else {
            // set the return content of this API method,
            $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                ->setMessage('oa.status.request.list.missing');
        }


        // set the content type
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);

        return $content;
    }

    /**
     * Retrieve an indicator if the method requires authentication before it may be executed
     *
     * @return bool
     */
    public function doesRequireAuthentication()
    {
        return true;
    }
}
```

```php
<?php
/**
 * OAMobileImageAPIMethod.php
 *
 * @copyright   Copyright (c) 2007-2013 Solware B.V. (http://www.solware.nl)
 */

class OAMobileImageAPIMethod extends Sef_Rest_Method_MethodAbstract
{
    /**
     * Retrieve an indicator if this method should response to the given request or not.
     *
     * @param Sef_Rest_Request $request
     * @return bool
     */
    public function shouldRespondToRequest(Sef_Rest_Request $request)
    {
        // build the request url for this method
        $requestUrl = sprintf('/^(%1$sgetImage(\/{0,1}))+$/', preg_quote($this->getService()->getBaseUrl(), '/'));

        // validate if the method type and request matches
        return $request->getMethodType() == Sef_Rest_Request::METHOD_POST &&
        preg_match($requestUrl, $request->getRequestUrl());
    }

    /**
     * Retrieve the Sef_Rest_Content_JsonSendContent object for this method.
     *
     * @param Sef_Rest_Request $request
     * @return Sef_Rest_Content_JsonSendContent
     */
    public function getContentByRequest(Sef_Rest_Request $request)
    {
        // create a new REST JSend return content object
        $content = new Sef_Rest_Content_JsonSendContent();

        // get the type of image to load
        $imageType = $request->getPostParameter('type', null);
        $objectId = $request->getPostParameter('id', null);

        // reset the json content type
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);

        // when the image type and id have been set try to load that image
        if ($imageType !== null && $objectId !== null) {

            try {
                // create the vfs file that will be fetched by the switch statement
                $vfsFile = null;

                switch ($imageType) {

                    // retrieves the file for the given location ID
                    case 'Location':
                        $vfsFile = OM_Location::getById($objectId)->getVfsFile();
                        break;

                    // loads the image for the given space
                    case 'Space':
                        $vfsFile = OM_Space::getById($objectId)->getVfsFile();
                        break;

                    // load the image for the given element
                    case 'Element':
                        $vfsFile = OM_Element::getById($objectId)->getVfsFile();
                        break;

                    // loads the image for the given defect
                    case 'Defect':
                        $vfsFile = OM_Defect::getById($objectId)->getVfsFile();
                        break;
                }

                if ($vfsFile !== null) {

                    // create new file system with the vfs file namespace and output this file
                    $vfsFileManager = new Filesystem($vfsFile->getVfsNamespace()->getNamespace());
                    $vfsFileManager->outputFile($vfsFile->getVfsFileId(), false);

                    // stop executing
                    die();
                }
            } catch (LoadException $exception) {

                // catch any load exception to set the error message
                $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                    ->setMessage('oa.image.request.unavailable');
            }
        } else {

            // set the return content of this API method,
            $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_ERROR)
                ->setMessage('oa.image.request.missing.parameters');
        }

        // return the created content
        return $content;
    }

     /**
     * Returns whether authentication is required for this API method
     *
     * @return bool
     */
    public function doesRequireAuthentication()
    {
        return false;
    }
}
```

```php
<?php
/**
 * OMMobileSynchronizeDefectsAPIMethod
 *
 * Use basic authentication to authorize and receive a unique access token.
 *
 * default call:
 * /api/v1/synchronizeDefects/
 *
 */
class OAMobileSynchronizeDefectsAPIMethod extends Sef_Rest_Method_MethodAbstract
{
    /**
     * Retrieve an indicator if this method should response to the given request or not.
     *
     * @param Sef_Rest_Request $request
     * @return bool
     */
    public function shouldRespondToRequest(Sef_Rest_Request $request)
    {
        // set the request type to html
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_HTML);

        // build the request url for this method
        $requestUrl = sprintf('/^(%1$ssynchronizeDefects(\/{0,1}))+$/', preg_quote($this->getService()->getBaseUrl(), '/'));

        // validate if the method type and request matches
        return $request->getMethodType() == Sef_Rest_Request::METHOD_GET &&
        preg_match($requestUrl, $request->getRequestUrl());
    }

    /**
     * Retrieve the Sef_Rest_Content_JsonSendContent object for this method.
     *
     * @param Sef_Rest_Request $request
     * @return Sef_Rest_Content_JsonSendContent
     */
    public function getContentByRequest(Sef_Rest_Request $request)
    {
        // create a new REST JSend return content object
        $content = new Sef_Rest_Content_JsonSendContent();

        // retrieve all locations, elements, spaces en defects
        $omLocations = OM_Location::getAll();
        $omSpaces = OM_Space::getAll();
        $omElements = OM_Element::getAll();
        $omDefects = OM_Defect::getAll();

        // process each synchronized element and add it to the array
        $JSONData = array
        (
            'locations'     => array(),
            'spaces'        => array(),
            'elements'      => array(),
            'defects'       => array()
        );

        // process locations
        for ($i = 0; $i < count($omLocations); $i++) {
            $JSONData['locations'][] = array
            (
                'id'            => $omLocations[$i]->getOmLocationId(),
                'description'   => $omLocations[$i]->getDescription()
            );
        }

        // process spaces
        for ($i = 0; $i < count($omSpaces); $i++) {
            $JSONData['spaces'][] = array(
                'id' => $omSpaces[$i]->getOmSpaceId(),
                'description' => $omSpaces[$i]->getDescription(),
                'om_location_id' => $omSpaces[$i]->getOmLocationId());
        }

        // process the elements
        for ($i = 0; $i < count($omElements); $i++) {
            $JSONData['elements'][] = array(
                'id' => $omElements[$i]->getOmElementId(),
                'description' => $omElements[$i]->getDescription(),
                'om_space_id' => $omElements[$i]->getOmSpaceId());
        }

        // process all defects
        for ($i = 0; $i < count($omDefects); $i++) {
            $JSONData['defects'][] = array(
                'id' => $omDefects[$i]->getOmDefectId(),
                'description' => $omDefects[$i]->getDescription(),
                'om_element_id' => $omDefects[$i]->getOmElementId());
        }

        // set the return content of this API method,
        $content->setStatus(Sef_Rest_Content_JsonSendContent::STATUS_SUCCESS)
            ->setCode(Sef_Rest_Response::STATUS_CODE_OK)
            ->setData($JSONData);

        // return the created content
        $request->setContentType(Sef_Rest_Request::CONTENT_TYPE_JSON);
        return $content;
    }

    /**
     * Retrieve an indicator if the method requires authentication before it may be executed
     *
     * @return bool
     */
    public function doesRequireAuthentication()
    {
        return true;
    }
}
```

```php
<?php
/**
 * Created by JetBrains PhpStorm.
 * User: stephandebakker
 * Date: 31-07-13
 * Time: 10:39
 * To change this template use File | Settings | File Templates.
 */

class OM_AddressLookUp
{
    // api call return values constants
    const APIResponseExceptionId            =   'exceptionId';
    const APIResponseAddressNotFound        =   'PostcodeNl_Service_PostcodeAddress_AddressNotFoundException';
    const APIResponseInvalidPassword        =   'PostcodeNl_Controller_Plugin_HttpBasicAuthentication_PasswordNotCorrectException';

    // the error array for validation errors
    private $validationError;
    const APIAddressErrorCode               =   'errorCode';
    const APIAddressErrorDescription        =   'errorDescription';

    /*
     *  Initialize the address lookup object
     */
    public function __construct()
    {
        // init validation array
        $this->validationError = array();
    }

    /*
     * Validates the given address information via a cache, when not found the postcode API will be referenced
     *
     * @param   string  zipcode
     * @param   string  houseNumber
     * @param   string  numberAddition
     * @return  OM_MobileLocationCache | false
     */
    public function validateAddress($zipcode, $houseNumber, $numberAddition = null)
    {
        // first check wether the address is already in the cache
        $cachedLocation = $this->validateAddressInCache($zipcode, $houseNumber, $numberAddition);

        // when a cached location has been found return the results immediatly
        if ($cachedLocation !== null) {

            // when the cached location is valid return the found address
            if ($cachedLocation->getIsValidAddress() === '1') {
                return $cachedLocation;
            }
            else
            {
                $this->validationError[self::APIAddressErrorCode] = -2;
                $this->validationError[self::APIAddressErrorDescription] = 'oa.address.validation.not_found';
                return null;
            }
        }
        else
        {
            // validate the address info using the api.postcode.nl
            return ($this->validateAddressInAPI($zipcode, $houseNumber, $numberAddition));
        }
    }

    /*
     * Private method that checks wether the given address already exists and is valid in the cached location table
     * @param string    zipcode
     * @param string    houseNumber
     * @param string    numberAddition
     * @return OM_Location | false
     */
    private function validateAddressInCache($zipcode, $houseNumber, $numberAddition = null)
    {
        // get the cached location from the database
        $omMobileLocationCache = new OM_MobileLocationCache();

        // return the result of the location cache
        return $omMobileLocationCache->getByZipCodeAndHouseNumber($zipcode, $houseNumber, false);
    }

    /*
     * Private method that initializes a connection to the api.postcode.nl server and validates the address info
     *
     * @param   string  APIURL
     * @return  OM_Location
     */
    private function validateAddressInAPI($zipcode, $houseNumber, $numberAddition = null)
    {
        // initialize the curl object
        $cURLConnection = curl_init(sprintf('%1$s%2$s/%3$s/%4$s', Config::get(Config::POSTCODE_API_URL), $zipcode, $houseNumber, $numberAddition));

        // configure the cURL object, set the credentials to HTTP authentication
        curl_setopt($cURLConnection, CURLOPT_USERPWD, sprintf('%1$s:%2$s', Config::get(Config::POSTCODE_API_USERNAME),
Config::get(Config::POSTCODE_API_PASSWORD)));
        curl_setopt($cURLConnection, CURLOPT_HEADER, false);
        curl_setopt($cURLConnection, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
        curl_setopt($cURLConnection, CURLOPT_RETURNTRANSFER, true);

        // execute the curl connectionand and parse the data as JSON
        $response = json_decode(curl_exec($cURLConnection), true);

        // get info about the just executed connection and free up the memory for this connection
        $statusCode = curl_getinfo($cURLConnection, CURLINFO_HTTP_CODE);
        curl_close($cURLConnection);

        // when the page not found status code is returned the address could not be found
        if ($statusCode != '200') {

            // when 404 is returned the address could not be found, insert the invalid address
            if ($statusCode == '404') {
                // insert the invalid address
                $location = new OM_MobileLocationCache();
                $location->setZipcode($zipcode);
                $location->setCachedDate(Sef_Helpers_DateTime::getSqlDateByTimestamp(time()));
                $location->setHouseNumber($houseNumber);
                $location->setHouseNumberAddition($numberAddition);
                $location->setIsValidAddress(false);

                // insert the location
                $location->insert();
```

```
            }
            // set default error code and description
            $this->validationError[self::APIAddressErrorCode] = -1;
            $this->validationError[self::APIAddressErrorDescription] = 'oa.address.validation.unknown';

            // check for the exception key to be set
            if (is_array($response) && isset($response[self::APIResponseExceptionId])) {
                // check the exception code to respond correctly, sets the error code and description
                switch ($response[self::APIResponseExceptionId])
                {
                    // when address not found
                    case self::APIResponseAddressNotFound:
                        $this->validationError[self::APIAddressErrorCode] = -2;
                        $this->validationError[self::APIAddressErrorDescription] = 'oa.address.validation.not_found';
                        break;

                    case self::APIResponseInvalidPassword:
                        $this->validationError[self::APIAddressErrorCode] = -3;
                        $this->validationError[self::APIAddressErrorDescription] = 'oa.address.validation.invalid_account';
                        break;
                }
            }

            return null;
        }
        else
        {
            // create the location object from the fetched location
            try {
                // create the location object
                $location = new OM_MobileLocationCache();

                $location->setCity($response['city']);
                $location->setStreet($response['street']);
                $location->setCachedDate(Sef_Helpers_DateTime::getSqlDateByTimestamp(time()));
                $location->setIsValidAddress(true);
                $location->setLatitude($response['latitude']);
                $location->setLongitude($response['longitude']);
                $location->setZipcode($zipcode);
                $location->setHouseNumber($houseNumber);
                $location->setHouseNumberAddition($numberAddition);

                // insert the cached location
                $location->insert();
                return $location;
            }
            catch (Exception $exception) {
                // when an exception is caught not all values have been set to create the location
                $this->validationError[self::APIAddressErrorCode] = -3;
                $this->validationError[self::APIAddressErrorDescription] = 'oa.address.validation.format_error';
                return null;
            }
        }
    }

    /*
     * Return the error object
     * @return  array
     */
    public function getValidationError()
    {
        return $this->validationError;
    }
}
```

```php
<?php
/**
 * Created by JetBrains PhpStorm.
 * User: stephandebakker
 * Date: 31-07-13
 * Time: 11:30
 * To change this template use File | Settings | File Templates.
 */

class OM_AddressLookUpLocation
{
    // constants indicating the response keys of the location object
    const OMLocationStreet                  =   'street';
    const OMLocationHouseNumber             =   'houseNumber';
    const OMLocationHouseNumberAddition     =   'houseNumberAddition';
    const OMLocationZipcode                 =   'postcode';
    const OMLocationCity                    =   'city';
    const OMLocationMunicipality            =   'municipality';
    const OMLocationProvince                =   'province';
    const OMLocationRDX                     =   'rdX';
    const OMLocationRDY                     =   'rdY';
    const OMLocationLatitude                =   'latitude';
    const OMLocationLongitude               =   'longitude';
    const OMLocationBAGNumberDesignationId  =   'bagNumberDesignationId';
    const OMLocationBAGAddressableObjectId  =   'bagAddressableObjectId';
    const OMLocationAddressType             =   'addressType';
    const OMLocationPurposes                =   'purposes';
    const OMLocationSurfaceArea             =   'surfaceArea';
    const OMLocationHouseNumberAdditions    =   'houseNumberAdditions';

    // the array with all validated values
    private $locationValues;

    /*
     * Creates a new LookUpLocation object with the provided data array
     */
    public function __construct($locationArray)
    {
        // check for all values to exist in the array, else cannot init this object
        if (isset(
            $locationArray[self::OMLocationStreet],
            $locationArray[self::OMLocationHouseNumber],
            $locationArray[self::OMLocationHouseNumberAddition],
            $locationArray[self::OMLocationZipcode],
            $locationArray[self::OMLocationCity],
            $locationArray[self::OMLocationMunicipality],
            $locationArray[self::OMLocationProvince],
            $locationArray[self::OMLocationRDX],
            $locationArray[self::OMLocationRDY],
            $locationArray[self::OMLocationLatitude],
            $locationArray[self::OMLocationLongitude],
            $locationArray[self::OMLocationBAGNumberDesignationId],
            $locationArray[self::OMLocationBAGAddressableObjectId],
            $locationArray[self::OMLocationAddressType],
            $locationArray[self::OMLocationPurposes],
            $locationArray[self::OMLocationSurfaceArea],
            $locationArray[self::OMLocationHouseNumberAdditions]))
        {
            // all values have been set so the object can be created
            $this->locationValues = $locationArray;
        }
        else {
            // not all expected values are set, return null object can not be created
            throw new InvalidArgumentException('Not all values have been set.');
        }
    }

    /*
     * Returns the associated value for a given key
     * @param   string(OMLocation constant) key
     * @return  string | null
     */
    public function locationValueForKey($key)
    {
        // when the key exists return its value
        if (isset($this->locationValues[$key])) {
            return $this->locationValues[$key];
        }
        else {
            // key does not exist return null
            return null;
        }
    }

    /*
     * returns a an array representation of the object
     * @return  array
     */
    public function toArray()
    {
        // return the representation array
        return $this->locationValues;
    }
}
```

```php
<?php

/**
 * OM_DayPart - A daypart on which a maintenance can be scheduled.
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_DayPart extends OM_DayPartBase
{
    /**
     * A static array with all available day parts in the database
     *
     * @var array (OM_DayPart)
     */
    private static $allOmDayParts = null;

    /*
     * Returns the time description as the day part
     * @return  string
     */
    public function __toString()
    {
        // get and return the time description
        return sprintf('%1$s', $this->getTimeDescription());
    }

    /**
     * Validates the given id in the day part array
     *
     * @param string $dayPartDescription
     * @return mixed
     */
    public static function getDayPartIdForDescription($dayPartDescription)
    {
        // check for the static array to be initialized
        if (self::$allOmDayParts === null) {
            self::$allOmDayParts = OM_DayPart::getAll();
        }

        // validate this day part id
        for ($i = 0; $i < count(self::$allOmDayParts); $i++) {
            if (strcmp($dayPartDescription, self::$allOmDayParts[$i]->getTimeDescription()) == 0) {
                return self::$allOmDayParts[$i]->getOmDayPartId();
            }
        }

        return false;
    }

    /**
     * Icon url for day part
     *
     * @return string
     */
    public static function getIconUrl()
    {
        return '/resources/images/icons/calendar-select.png';
    }
}
```

```php
<?php

/**
 * OM_Defect - The defect type that can be linked to a maintenance.
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_Defect extends OM_DefectBase
{
    /*
     * Returns a string description about the defect
     * @return  string
     */
    public function __toString()
    {
        // return the name of the defect
        return $this->getDescription();
    }
}
```

```php
<?php

/**
 * OM_Element - The element type that holds a couple f spaces
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_Element extends OM_ElementBase
{
    /*
     * Return the element description as the string representation
     * @return  string
     */
    public function __toString()
    {
        // return the description
        return $this->getDescription();
    }
}
```

```php
<?php

/**
 * OM_Location - The location type that can be linked to a maintenance.
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_Location extends OM_LocationBase
{
    /*
     * Returns the location description as the string
     * @return  string
     */
    public function __toString()
    {
        // return location description
        return $this->getDescription();
    }
}
```

```php
<?php

/**
 * OM_Maintenance - All maintenance objects posted by the mobile application users
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_Maintenance extends OM_MaintenanceBase
{
    // permissions available
    const PERMISSION_NEW    = 'om:maintenance_new';
    const PERMISSION_EDIT   = 'om:maintenance_edit';
    const PERMISSION_VIEW   = 'om:maintenance_view';
    const PERMISSION_REMOVE = 'om:maintenance_remove';

    /*
     * Returns a string representation of the maintenance object
     *
     * @return  string
     */
    public function __toString()
    {
        // create a limited description string, when more than 28 characters the description will be limited to 25 characters including a "..."
        //  to indicate that the description has more characters
        $defectDescription = sprintf('%1$s', Sef_Helpers_String::getDisplayStringWithMaxLength($this->getDescription(), 25));

        // add ... to indicate that the description has more characters
        if (strlen($defectDescription) == 25) {
            $defectDescription .= '...';
        }

        // return formatted string with defect and description when available
        if (strlen($defectDescription) > 0) {
            return sprintf('%1$s - "%2$s"', $this->getOmDefect(), $defectDescription);
        }
        else {
            // only return the defect string without description
            return $this->getOmDefect()->__toString();
        }
    }

    /**
     *  Return the link to download the image file for this maintenance
     */
    public function getDisplayOverviewMaintenanceFile()
    {
        if ($this->getVfsFile() !== null) {
            // create the file system object with the namespace of the vfs file
            return sprintf('<a href="/mod/om/maintenance/displayMaintenanceImage/?file=%1$s">%2$s</a>', $this->getVfsFileId(), $this->getVfsFile()->getName());
        }
        else {
            return __('om.maintenance.image.not.provided');
        }
    }

    /**
     * Return the maintenance object icon url
     *
     * @return string
     */
    public static function getIconUrl()
    {
        return '/resources/images/mod/om/icons/clipboard-text.png';
    }
}
```

```php
<?php

/**
 * OM_MaintenanceUser - The user who has sent the maintenance, contact and adress info.
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_MaintenanceUser extends OM_MaintenanceUserBase
{
    /**
     * Override the default sort of the maintenance user object
     *
     * @var array
     */
    protected $_defaultSort = array(
        self::CITY => self::ASCENDING,
        self::ADRESS => self::ASCENDING,
    );


    /**
     * Defines wether the state of the maintenance user has changed
     *
     * @var bool
     */
    private $maintenanceStateHasChanged = false;

    /**
     * Permission values
     */
    const PERMISSION_NEW        = 'om:maintenanceUser_new';
    const PERMISSION_EDIT       = 'om:maintenanceUser_edit';
    const PERMISSION_VIEW       = 'om:maintenanceUser_view';
    const PERMISSION_REMOVE     = 'om:maintenanceUser_remove';

    /**
     * Returns the string representation of the maintenance user database object
     *
     * @return string
     */
    public function __toString()
    {
        if ($this->getOmAvailability() !== null) {
            // return the formatted string
            return sprintf('%1$s %2$s - (%3$s)', $this->getAdress(), $this->getHouseNumber(), $this->getOmAvailability());
        } else {
            return sprintf('%1$s - %2$s', $this->getAdress(), $this->getHouseNumber());
        }
    }

    /**
     * Generates a table for the email content which describes all defects that have been uploaded for this reservation
     *
     * @return string
     */
    public function getDisplayMaintenanceTableOverview()
    {
        // retrieve all maintenance objects for this reservation
        $omMaintenance = OM_Maintenance::getAllWithSortByWhere(null,
            array(
                OM_Maintenance::OM_MAINTENANCE_USER_ID      =>  $this->getOmMaintenanceUserId(),
            ));

        // when no defects are available return a description string
        if (count($omMaintenance) == 0) {
            return __('om.maintenance.user.defect.overview.none');
        }

        // create the start html of the table
        $maintenanceTable = '<table border="0">';

        foreach ($omMaintenance as $currentOmMaintenance) {
            $maintenanceTable .= '<tr>';

            // add the row for the selected defect
            $maintenanceTable .= sprintf('<td>%1$s</td><td>%2$s</td>', __('om.maintenanceuser.table.defect.title'), $currentOmMaintenance-
>getOmDefect()->getDescription());

            // add the user formatted description string
            $maintenanceTable .= sprintf('<td>%1$s</td><td>%2$s</td>', __('om.maintenanceuser.table.description.title'), $currentOmMaintenance-
>getDescription());

            // add whether the user has provided an image and close the current row
            $maintenanceTable .= sprintf('<td>%1$s</td><td>%2$s</td>', __('om.maintenanceuser.table.image.title'), ($currentOmMaintenance-
>getVfsFileId() !== null) ? 'Ja' : 'Nee');
            $maintenanceTable .= '</tr>';
        }

        // close the table tag and return the created data
        $maintenanceTable .= '</table>';
        return $maintenanceTable;
    }

    /**
     * Processes this maintenance to the approved state and sends this user the approved email, throws exception when the
     * state was already in approved.
     *
     * @throws Exception
     */
    public function processMaintenanceUserToApprovedState()
    {
        // check whether the current state is not equal to the confirmed state
        if ($this->getOmMaintenanceStateId() != OM_MaintenanceState::CONFIRMED_STATE) {
            // set the new state
            $this->setOmMaintenanceStateId(OM_MaintenanceState::CONFIRMED_STATE);

            // send the user a confirmation email that the reservation is approved
            $omMaintenanceUserApprovedMessage = new OM_MaintenanceUserApprovedMessage($this);
            $omMaintenanceUserApprovedMessage->send();

            // create new history object
            $omMaintenanceUserStateHistory = new OM_MaintenanceUserStateHistory();
            $omMaintenanceUserStateHistory->setOmMaintenanceStateId(OM_MAintenanceState::CONFIRMED_STATE);
            $omMaintenanceUserStateHistory->setStateChangedDate(Sef_Helpers_DateTime::getSqlDateTimeByTimestamp(time()));
            $omMaintenanceUserStateHistory->setStateCorrespondingEmailWasSent(true);
            $omMaintenanceUserStateHistory->setOmMaintenanceUserId($this->getOmMaintenanceUserId());

            // save the history object
            $omMaintenanceUserStateHistory->save();
        } else {
```

```
            // throw exception and specify why the exception was thrown
            throw new Exception(sprintf(__('om.maintenanceuser.approve.error.already.approved'), $this->getDisplayOverviewAddress()));
        }
    }

    /**
     * Returns the date description of the availability object for the overview of maintenance users
     *
     * @return string
     */
    public function getDisplayOverviewOmDayPartDate()
    {
        if ($this->getOmAvailability() !== null) {
            // return the date formatted string for the overview table
            return Sef_Helpers_DateTime::getDisplayDateTimeByDate($this->getOmAvailability()->getDate());
        } else {
            return '';
        }
    }


    /**
     * Retrieves the specific action buttons for this maintenance user
     *
     * @return string
     */
    public function getDisplayActionButtons()
    {
        // return the three buttons that allows the reporting as spam, show all defects uploaded for this maintenance and
        // the button that invalidates the maintenance
        return sprintf('<a href="/mod/om/maintenanceUser/setMaintenanceArchived/%1$s/"><img width = "16" height = "16" src = "/resources/images/
icons/vise-drawer.png"/></a>', $this->getOmMaintenanceUserId());
    }

    /**
     * Retrieve the icon url that is used to represent a maintenance user item
     *
     * @return string Location of the icon that represents a work order item
     */
    public static function getIconUrl()
    {
        return '/resources/images/mod/om/icons/clipboard-text.png';
    }

    public function getOmMaintenanceIdWithPrefixForArchive()
    {
        if ($this->getOmAvailability() !== null) {
            return sprintf('%1$s-%2$s', date('ym', strtotime($this->getOmAvailability()->getDate())), $this->getOmMaintenanceUserId());
        } else {
            return $this->getOmMaintenanceUserId();
        }
    }

    /**
     * Returns the link and image for the view item
     *
     * @return string
     */
    public function getOmMaintenanceIdColumnForAvailabilityOverviewFormConfig()
    {
        return sprintf('<a href="/mod/om/maintenanceUser/viewItem/%1$s/?""><img src="/resources/images/mod/om/icons/clipboard-text.png"/>%2$s-%3$s</
a>', $this->getOmMaintenanceUserId(), date('ym', strtotime($this->getOmAvailability()->getDate())), $this->getOmMaintenanceUserId());
    }

    /**
     * Return the om maintenance id with the prefix included
     *
     * @return string
     */
    public function getOmMaintenanceIdWithPrefix()
    {
        if ($this->getOmAvailability() !== null) {
            // return the id with prefix
            return sprintf('<a href="/mod/om/maintenanceUser/viewItem/%1$s/">%2$s-%3$s</a>',$this->getOmMaintenanceUserId(), date('ym',
strtotime($this->getOmAvailability()->getDate())), $this->getOmMaintenanceUserId());
        } else {
            return sprintf('<a href="/mod/om/maintenanceUser/viewItem/%1$s/">%2$s</a>',$this->getOmMaintenanceUserId(), $this-
>getOmMaintenanceUserId());
        }
    }

    /**
     * Return the address information in a single field for the overview
     *
     * @return string
     */
    public function getDisplayOverviewAddress()
    {
        // format the address string
        return sprintf('%1$s %2$s%3$s', $this->getAdress(), $this->getHouseNumber(), $this->getAdressAddition());
    }
}
```

```php
<?php

/**
 * OM_Space - The defect type that can be linked to a maintenance.
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_Space extends OM_SpaceBase
{
    public function __toString()
    {
        return $this->getDescription();
    }
}
```

```php
<?php

/**
 * OM_UuidAuthentication - A single row represents a mobile device with token to get authenticated access for usage in the API.
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_UuidAuthentication extends OM_UuidAuthenticationBase implements Sef_Rest_Authentication_AccountInterface
{
    /**
     * An array of key value objects declared for settings
     *
     * @var array
     */
    private $settings = array();

    /**
     * Retrieve a setting for the current user, if no matching setting was found the default value will be returned.
     *
     * @param string $setting
     * @param mixed  $defaultValue (Optional) Defaults to null.
     * @return null|string
     */
    public function getSetting($setting, $defaultValue = null)
    {
        if (array_key_exists($setting, $this->settings)) {

            // return the value for the given key
            return $this->settings[$setting];

        } else {
            // the setting key was not found, return null
            return null;
        }
    }

    /**
     * Assign a setting for the current user object.
     *
     * @param string $setting
     * @param mixed  $value
     * @return Sef_Rest_Authentication_AccountInterface
     */
    public function setSetting($setting, $value)
    {
        // set the provided value for the setting key
        $this->settings[$setting] = $value;
    }


    /**
     * Returns whether the provided credentials can be authenticated, this means that the password will be validated against
     * a hashed version of this string.
     *
     * @param string $username
     * @param string $password
     * @return bool
     */
    public static function canAuthenticateWithUsernameAndPassword($username, $password)
    {
        // a UUID must be exactly 64 characters in length, when not return false because this password cannot be authenticated
        if (!is_string($password) || strlen($password) !== 64) {
            return false;
        }

        // fetches the UUID authentication object that represents the current username
        $uuidAuthentication = OM_UuidAuthentication::getAllWithSortByWhere(null, array(
            OM_UuidAuthentication::UUID_USERNAME    =>  $username,
            OM_UuidAuthentication::UUID_LOGIN_TOKEN =>  $password,
        ), 1);

        // check for the result to be an array and is exactly 1 object
        if (is_array($uuidAuthentication) && count($uuidAuthentication) === 1) {
            return true;
        } else {
            return false;
        }
    }

    /**
     * Returns a found authentication user for the mobile API of the BouwCloud OA application
     *
     * @param string $username
     * @return OM_UuidAuthentication|null
     */
    public static function getByUsername($username)
    {
        // process the username by stripping al special characters from the string
        $processedUsername = Sef_Helpers_String::convertToSimpleString($username, true, true, true);

        $uuidAuthenticationUser = self::getAllWithSortByWhere(null, array(
            OM_UuidAuthentication::UUID_USERNAME    =>  $processedUsername,
        ), 1);

        // when the result is an array return the first object in this array, else return null
        if (is_array($uuidAuthenticationUser) && count($uuidAuthenticationUser) === 1) {
            return $uuidAuthenticationUser[0];
        } else {
            return null;
        }
    }

    /**
     * Find the only object for the provided password
     *
     * @param $password
     * @param bool $assertOnFound
     * @return OM_UuidAuthentication|null
     * @throws Exception|LoadException
     */
    public static function getFirstByUuidPassword($password, $assertOnFound = true)
    {
        try {
            // get all authentication objects with a limit of 1 for the provided password
            $authenticationObjects = OM_UuidAuthentication::getAllWithSortByWhere(null, array(
                OM_UuidAuthentication::UUID_PASSWORD    =>  $password,
            ), 1);

            // verify that a single object has been found
            if (count($authenticationObjects) === 1) {
```

```
                return $authenticationObjects[0];
            } else {
                return null;
            }

        } catch (LoadException $exception) {

            // rethrow exception when needed to assert
            if ($assertOnFound) {
                throw $exception;
            } else {

                // otherwise just return null
                return null;
            }
        }
    }

    /**
     * Returns whether the current authenticated user has rights to edit content or remove the maintenance user with the
     * provided maintenance user id
     *
     * @param int $omMaintenanceUserId
     * @return bool
     */
    public function hasAccessToMaintenanceUserWithId($omMaintenanceUserId)
    {
        // query the database to verify whether the authenticated account has access to the provided maintenance user
        $accessCount = OM_MaintenanceUser::getCountAllByWhere(array(
            OM_MaintenanceUser::OM_MAINTENANCE_USER_ID      =>  $omMaintenanceUserId,
            OM_MaintenanceUser::OM_UUID_AUTHENTICATION_ID   =>  $this->getOmUuidAuthenticationId(),
        ));

        // when there is exactly one object found the user has access to the maintenance user object
        return ($accessCount === 1);
    }
}
```

```php
<?php
/**
 * UuidAuthenticationProvider.php
 *
 * @copyright   Copyright (c) 2007-2013 Solware B.V. (http://www.solware.nl)
 */

class UuidAuthenticationProvider extends  Sef_Rest_Authentication_AuthenticationProviderAbstract
{
    /**
     * Retrieve an indicator if authorization took place
     *
     * @param Sef_Rest_Request $request
     * @return bool
     */
    public function isAuthorized(Sef_Rest_Request $request)
    {
        // only for the authentication request basic auth is not needed
        if (strcmp($request->getRequestUrl(), sprintf('%1$sauthenticationRequest/', $this->getService()->getBaseUrl())) == 0) {
            return false;
        }

        // when the username and password strings were not set authentication cannot occur
        if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW'])) {

            return false;
        }

        // authenticate the user with the provided username and password
        $auth = OM_UuidAuthentication::canAuthenticateWithUsernameAndPassword($_SERVER['PHP_AUTH_USER'], $_SERVER['PHP_AUTH_PW']);

        return $auth;
    }

    /**
     * Assign the current logged in account and return the instance that was assigned (or null if nothing was assigned)
     *
     * @param Sef_Rest_Request $request
     * @return null|Sef_Rest_Authentication_AccountInterface
     */
    public function setAuthorizedAccountFromRequest(Sef_Rest_Request $request)
    {
        // only for the authentication request basic auth is not needed
        if (strcmp($request->getRequestUrl(), sprintf('%1$sauthenticationRequest/', $this->getService()->getBaseUrl())) == 0) {
            return null;
        }

        // assign the default value
        $authorizedAccount = null;

        // validate if the request authorized
        if ($this->isAuthorized($request) === true) {

            // extract a User object for the given username
            $authorizedAccount = OM_UuidAuthentication::getByUsername($_SERVER['PHP_AUTH_USER'], false);
        }

        // return the authorized account or null on failure
        return $authorizedAccount;
    }
}
```

```php
<?php

/**
 * OM_AvailabilityController - A record describes the amount of available employees at a certain date and day part.
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_AvailabilityController extends OM_AvailabilityControllerBase
{
    /**
     * Crud config
     *
     * @var array
     */
    protected $_config = array(
        self::CONFIG_ENTITY_NAME            => OM_Availability::TABLE,
        self::CONFIG_BASE_LANGUAGE_TAG      => 'om.availability',
        self::CONFIG_FORM_NAME_EXTENDED     => 'OM_AvailabilityForm',
        self::CONFIG_FORM_NAME_OVERVIEW     => 'OM_AvailabilityOverviewForm',
        self::CONFIG_PERMISSION_EDIT        => OM_Availability::PERMISSION_EDIT,
        self::CONFIG_PERMISSION_NEW         => OM_Availability::PERMISSION_NEW,
        self::CONFIG_PERMISSION_REMOVE      => OM_Availability::PERMISSION_REMOVE,
        self::CONFIG_PERMISSION_VIEW        => OM_Availability::PERMISSION_VIEW,

        parent::CONFIG_EXTENDED_TAB_COLOR   => 'green',
        parent::CONFIG_EXTENDED_TABS        => array()
    );

    /**
     * Retrieve additional custom menu items at the overview
     *
     * @return array
     */
    protected function getAdditionalOverviewMenuItems()
    {
        $menuItems = array();

        // download status history
        $menuItems[] = array(
            'href'      => sprintf('%1$sdownloadAvailabilityExcelTemplate', $this->getBaseRequest()),
            'label'     => 'om.availability.excel.template.export',
            'img'       => '/resources/images/icons/document-excel.png',
            'requires'  => OM_WorkOrder::PERMISSION_VIEW,
        );

        // download status history
        $menuItems[] = array(
            'href'      => sprintf('%1$suploadAvailabilityExcelDocument', $this->getBaseRequest()),
            'label'     => 'om.availability.excel.template.import',
            'img'       => '/resources/images/icons/document-excel.png',
            'requires'  => OM_WorkOrder::PERMISSION_VIEW,
        );

        return $menuItems;
    }

    /**
     * Override the display view item action to show all reservation objects for this availability
     *
     * @param DataObject $entityObject
     * @param array $menuItemsConfig
     * @param array $tabItemsConfig
     * @param array|Form $forms
     * @param array $extraOptions (Optional) Default null
     */
    protected function _displayViewItemAction(DataObject $entityObject, array $menuItemsConfig = null, array $tabItemsConfig = null, $forms = null,
array $extraOptions = null)
    {
        if ($entityObject instanceof OM_Availability) {

            // get the related availability to maintenance user form
            $omMaintenanceUserForm = new OM_AvailabilityMaintenanceUserOverviewForm(array($entityObject,
'getAllRelatedOmMaintenanceuserThruOmAvailabilityId'), array($entityObject, 'getCountAllRelatedOmMaintenanceuserThruOmAvailabilityId'), null,
Form::DISPLAY_MODE_VIEW);
            $omMaintenanceUserForm->setLabel(__('om.maintenanceuser.viewitem.all.maintenances.title'));

            if (is_array($forms)) {
                $forms[] = $omMaintenanceUserForm;
            } else {
                $forms = array($forms, $omMaintenanceUserForm);
            }
        }

        // return the parent result
        parent::_displayViewItemAction($entityObject, $menuItemsConfig, $tabItemsConfig, $forms, $extraOptions);
    }

    /**
     * Method for importing new availability objects in the database, generates errors on failure.
     *
     * @return string
     */
    public function uploadAvailabilityExcelDocument()
    {
        // create the form that shows the excel file upload
        $excelUploadForm = new OM_ImportAvailabilityFileForm('Import voor: Beschikbaarheid mobiele applicatie');
        $errors = array();

        // check for the form to be posted and is still valid
        if ($excelUploadForm->isPosted() && $excelUploadForm->isValid()) {

            // get the file location of the newly uploaded excel file and create the PHPExcel object for this file
            $fileLocation    = $excelUploadForm->getControlByName('import_file')->getValue();
            $overwriteOption = $excelUploadForm->getControlByName('should_overwrite_field')->getValue();
            $shouldOverwriteExistingAvailability = false;

            // when the option to overwrite availability is checked set the boolean
            if (!empty($overwriteOption)) {
                $shouldOverwriteExistingAvailability = true;
            }

            // create a new object representation of the uploaded Excel file
            $excelFile = new ExcelImport($fileLocation);

            // check for a minimum of 1 worksheet
            if ($excelFile->getCountWorkSheets() > 0) {

                // retrieve all sheet names to get a valid int to fetch the rows
                foreach ($excelFile->getSheetNames() as $sheetId => $sheetName) {
                    $sheetRows = $excelFile->getAllRows($sheetId);
```

```
                    // process all rows
                    foreach ($sheetRows as $currentIndex => $currentRow) {

                        // skip the first row (this is the title row)
                        if ($currentIndex == 0) {
                            continue;
                        }

                        // validate the column count
                        if (count($currentRow) == 4) {
                            // validate the values of this row
                            $employeeCount = (int)($currentRow[3]);
                            $currentDayPartDescription = $currentRow[2];
                            $currentTimeStamp = strtotime($currentRow[1]);

                            if (!$currentTimeStamp) {
                                $errors[] = sprintf(__('om.availability.import.date.invalid'), $currentRow[0]);
                                continue;
                            } else {
                                $currentDate = date('Y-m-d', $currentTimeStamp);
                            }

                            // validate the description of the day part time string
                            $currentDayPart = OM_DayPart::getDayPartIdForDescription($currentDayPartDescription);

                            // validate for the day part to be found
                            if ($currentDayPart === false) {
                                $errors[] = sprintf(__('om.availability.import.daypart.invalid'), $currentDayPartDescription);
                                continue;
                            }

                            // create the availability to insert or update
                            $currentOmAvailability = new OM_Availability();
                            $currentOmAvailability->setDate($currentDate);
                            $currentOmAvailability->setOmDayPartId($currentDayPart);
                            $currentOmAvailability->setAvailabilityCount($employeeCount);

                            // set error message to null
                            $errorMessage = null;

                            // try to save or update the availability
                            if (OM_Availability::saveImportedAvailabilityWithErrorOnDuplicate($currentOmAvailability,
$shouldOverwriteExistingAvailability, $errorMessage) === false) {
                                $errors[] = $errorMessage;
                            }

                            // increase row counter
                            $currentRow++;
                        } else {
                            // add error there must be exactly 4 cells in the row
                            $errors[] = sprintf(__('om.availability.import.row.invalid'), $currentRow);
                        }
                    }
                }
            }
        }

        // get the menu items
        $menuItemsConfig = array(
            array(
                'href'     => $this->getBaseRequest() . 'availability',
                'label'    => 'core.back',
                'img'      => '/resources/images/icons/arrow-180.png',
                'requires' => OM_Availability::PERMISSION_VIEW
            ),
        );

        // when the error array has input show them
        if (count($errors) >= 1) {
            // generate a list of errors
            $errorList = '<ul>';

            // add each error
            foreach ($errors as $currentError) {
                $errorList .= sprintf('<li>%1$s</li>', $currentError);
            }

            // show the error list
            $errorList .= '</ul>';
            $this->getView()->setFlashMessageDirect($errorList, Sef_Routing_Mvc_View::FLASHMESSAGE_TYPE_ERROR);
        } else if ($excelUploadForm->isPosted() && $excelUploadForm->isValid()) {
            // successfully imported all availabilities
            $this->getView()->setFlashMessageDirect(__('om.availability.import.success'), Sef_Routing_Mvc_View::FLASHMESSAGE_TYPE_SUCCESS);
        }

        // show the form
        return Sef_Helpers_Layout::displayDefaultTemplate($menuItemsConfig, null, $excelUploadForm);
    }

    /**
     * This method generates the template Excel file for the upcoming days from the last available day
     * The Excel download will be written to the php output
     */
    public function downloadAvailabilityExcelTemplate()
    {
        // create the new PHPExcel object and generate the active sheet
        $excelTemplate = new ExcelExport('Beschikbaarheid template');
        $activeSheet = $excelTemplate->createSheet(null, 'Template');

        // retrieve the current date and set date locale to dutch
        $currentTime = time();
        setlocale(LC_ALL, 'nl_NL');

        // set the current cell values
        $currentRow = 2;

        // fetch all day parts to insert in the excel document
        $allDayParts = OM_DayPart::getAll();

        // set the title for the columns as the first cells
        $activeSheet->setCellValue('A1', __('om.availability.excel.date.description.title'));
        $activeSheet->setCellValue('B1', __('om.availability.excel.date.title'));
        $activeSheet->setCellValue('C1', __('om.availability.excel.daypart.title'));
        $activeSheet->setCellValue('D1', __('om.availability.excel.employees.title'));

        for ($i = 0; $i < 31; $i++) {
            // when no weekday is found add it to the Excel document
            if (!Sef_Helpers_DateTime::isWeekendByTimestamp($currentTime)) {
```

```
                foreach ($allDayParts as $currentDayPart) {
                    // set values of cells
                    $activeSheet->setCellValue('A'. $currentRow, strftime('%A, %e %B %Y', $currentTime));
                    $activeSheet->setCellValue('B'. $currentRow, date('Y-m-d', $currentTime));
                    $activeSheet->setCellValue('C'. $currentRow, $currentDayPart->getTimeDescription());
                    $activeSheet->setCellValue('D'. $currentRow, $currentDayPart->getDefaultEmployeeCount());

                    // increase the row counter
                    $currentRow++;
                }
            }

            // increase the time counter by a single day
            $currentTime += 86400;
        }

        // We'll be outputting an excel file
        $activeSheet->setAutoSizeOnColumns(0, 1);
        $excelTemplate->outputAs2007('Template.xlsx', true);
    }
}
```

```php
<?php

/**
 * OM_AvailabilityForm
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_AvailabilityForm extends OM_AvailabilityFormBase
{
    /**
     * Retrieve array of field which needs to be rendered
     *
     * @return array
     */
    protected function getFieldsForForm()
    {
        return array(
            OM_AvailabilityBase::DATE,
            OM_AvailabilityBase::OM_DAY_PART_ID,
            OM_AvailabilityBase::AVAILABILITY_COUNT,
        );
    }

    /**
     * Override the save method, when the dayart and date already exist show an error message
     *
     * @return bool
     */
    public function save()
    {
        // fill data of the object
        $this->getOmAvailability()->fill($this->getPostData());

        // the new availability
        $omAvailability = $this->getOmAvailability();

        if ($omAvailability->isNew()) {
            // when the object is already existing in the database throw a new excpetion
            if ($omAvailability->exists(array(
                OM_Availability::DATE               =>  $omAvailability->getDate(),
                OM_Availability::OM_DAY_PART_ID     =>  $omAvailability->getOmDayPartId(),
            ))) {
                $this->addErrorForItem(OM_Availability::DATE, 'Deze datum bestaat al in combinatie met het opgegeven dagdeel.');
                $this->addErrorForItem(OM_Availability::OM_DAY_PART_ID, 'Dit dagdeel bestaat al in combinatie met de opgegeven datum.');
                return false;
            }
        }

        // call save from parent
        return parent::save();
    }

    /**
     * Return which grouped fields are available in this form
     *
     * @return array
     */
    public function _getGroupedFieldsColumns()
    {
        $groups[] = array
        (
            self::GROUP_LABEL   => '',
            self::GROUP_COLUMN  => self::GROUP_COLUMN_LEFT,
            self::GROUP_ITEMS   => array(
                OM_Availability::DATE,
                OM_Availability::OM_DAY_PART_ID,
                OM_Availability::AVAILABILITY_COUNT,
            )
        );

        return $groups;
    }


}
```

```php
<?php

/**
 * OM_AvailabilityOverviewForm
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_AvailabilityOverviewForm extends OM_AvailabilityOverviewFormBase
{
    // the column identifier for actions
    const COLUMN_RESERVATION_COUNT     =   'actions';

    /**
     * Returns the default sort used for dates and day part
     *
     * @return array
     */
    protected function getSortDefault()
    {
        return array(
            OM_Availability::DATE => OM_Availability::ASCENDING,
            OM_Availability::OM_DAY_PART_ID.'.'.OM_DayPart::OM_DAY_PART_ID => OM_Availability::ASCENDING,
        );
    }

    /**
     * Retrieve fields for columns
     *
     * @return array
     */
    protected function getColumnsOverviewForm()
    {
        return array(
            OM_AvailabilityBase::DATE,
            OM_AvailabilityBase::OM_DAY_PART_ID.'.'.OM_DayPart::OM_DAY_PART_ID,
            OM_AvailabilityBase::AVAILABILITY_COUNT,
            self::COLUMN_RESERVATION_COUNT,
        );
    }

    /**
     * Retrieve column form config
     *
     * @return array
     */
    protected function getColumnOverviewFormConfig($columnName)
    {
        switch ($columnName) {
            case self::COLUMN_RESERVATION_COUNT:
                return $this->getTotalReservationCountOverviewFormconfig();
                break;
            case OM_AvailabilityBase::OM_DAY_PART_ID.'.'.OM_DayPart::OM_DAY_PART_ID:
                return $this->getOmDayPartIdColumnOverviewFormConfig();
                break;
            case OM_AvailabilityBase::AVAILABILITY_COUNT:
                return $this->getAvailabilityCountColumnOverviewFormConfig();
                break;
            case OM_AvailabilityBase::DATE:
                return $this->getDateColumnOverviewFormConfig();
                break;
        };
    }

    /**
     * Override the default configuration for the day part column
     *
     * @return array
     */
    public function getOmDayPartIdColumnOverviewFormConfig()
    {
        return array(
            'sortField'     =>  OM_Availability::OM_DAY_PART_ID.'.'.OM_DayPart::OM_DAY_PART_ID,
            'label'         =>  'om.availability.daypart.title',
            'width'         =>  '*',
            'property'      =>  'getDisplayOverviewTimeDescription',
        );
    }

    /**
     * Return the specific date configuration column
     *
     * @return array
     */
    public function getDateColumnOverviewFormConfig()
    {
        return array(
            'sortField'     =>  OM_Availability::DATE,
            'label'         =>  'om.availability.date.title',
            'width'         =>  '*',
            'property'      =>  'getDisplayOverviewDate',
        );
    }

    /**
     * Override default column width
     *
     * @return array
     */
    public function getOverrideDefaultWidthColumns()
    {
        return array(
            OM_Availability::DATE                   =>  '400',
            OM_Availability::OM_DAY_PART_ID.'.'.OM_DayPart::OM_DAY_PART_ID     =>  '75',
            OM_Availability::AVAILABILITY_COUNT =>  '25',
            self::COLUMN_RESERVATION_COUNT      =>  '25'
        );
    }

    /**
     * Returns the action column for the table
     *
     * @return array
     */
    public function getTotalReservationCountOverviewFormconfig()
    {
        return array(
            'sortField' =>  null,
            'label'     =>  'Aantal reserveringen',
            'width'     =>  '*',
            'property'  =>  'getTotalReservationCount',
```

```
            'align'     =>  'right',
        );
    }

    /**
     * Override the default columns that are enabled for sorting
     *
     * @return array
     */
    public function getSortColumns()
    {
        return array(
            OM_AvailabilityBase::DATE,
            OM_AvailabilityBase::OM_DAY_PART_ID.'.'.OM_DayPart::OM_DAY_PART_ID,
            OM_AvailabilityBase::AVAILABILITY_COUNT,
        );
    }

    /**
     * Returns the overridden form configuration
     *
     * @return array
     */
    public function getFormConfig()
    {
        $formConfig = parent::getFormConfig();
        $formConfig['label'] = sprintf(__('om.availability.overview.label.title'), $this->getObjectsCount());

        // when the key does not exist of the
        $currentSorts = $this->getSessionData(self::SESSION_DATA_SORT);

        // add the default day part sorting when not added
        if (!array_key_exists(OM_Availability::OM_DAY_PART_ID, $currentSorts)) {
            $currentSorts[OM_Availability::OM_DAY_PART_ID.'.'.OM_DayPart::OM_DAY_PART_ID] = OM_Availability::ASCENDING;
            $this->setSessionData(self::SESSION_DATA_SORT, $currentSorts);
        }

        // get the amount of availability date objects that are sooner than the current date, this way
        // we can calculate which page number should be set initially
        $countSoonerDates = OM_Availability::getCountAllByWhere(array(
            '[<]'.OM_Availability::DATE      =>   Sef_Helpers_DateTime::getSqlDateByTimestamp(),
        ));

        // calculate the page to be shown with the current date
        if ($countSoonerDates <= $this->getDefaultPerPage()) {
            $this->setSessionData(self::SESSION_DATA_PAGE_NR, 1);
        } else {
            // calculate the page number to show the current dat in that page
            $newPageNR = floor($countSoonerDates / $this->getDefaultPerPage()) + 1;
            $this->setSessionData(self::SESSION_DATA_PAGE_NR, $newPageNR);
        }

        return $formConfig;
    }
}
```

```php
<?php

/**
 * OM_DayPartController - A daypart on which a maintenance can be scheduled.
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_DayPartController extends OM_DayPartControllerBase
{
}
```

```php
<?php

/**
 * OM_DayPartOverviewForm
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_DayPartOverviewForm extends OM_DayPartOverviewFormBase
{
    /**
     * Returns the overridden form configuration
     *
     * @return array
     */
    public function getFormConfig()
    {
        $formConfig = parent::getFormConfig();
        $formConfig['label'] = sprintf(__('om.daypart.overview.label.title'), $this->getObjectsCount());

        return $formConfig;
    }

    /**
     * Overrides the default columns width for the daypart overview table
     *
     * @return array
     */
    public function getOverrideDefaultWidthColumns()
    {
        return array(
            OM_DayPart::TIME_DESCRIPTION        =>  '*',
            OM_DayPart::START_TIME              =>  '100',
            OM_DayPart::END_TIME                =>  '100',
            OM_DayPart::DEFAULT_EMPLOYEE_COUNT  =>  '150',
        );
    }

}
```

```php
<?php

/**
 * OM_MaintenanceController - All maintenance objects posted by the mobile application users
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_MaintenanceController extends OM_MaintenanceControllerBase
{

    /*
   * This method can get called from the maintenance user controller, when pressed the user will be redirected to a page with all
   * of the added maintenances linked to that user
   */
    public function displayMaintenances()
    {
        // validate the permissions against the access controller.
        ACL_PermissionManager::checkAccessStrict($this->getCrudConfig(self::CONFIG_PERMISSION_VIEW));

        $menuItemsConfig    = array();

        // set history data
        $linkSuffix = '/?' . $this->getHistoryData()->generateForwardLinkSuffix($this->getDataSource());

        // setup menu items configuration which is in fact a link to create a new entity.
        if ($this->getCrudConfig(self::CONFIG_PERMISSION_NEW) !== null) {
            $menuItemsConfig[]  = array(
                'href'      => $this->getBaseRequest(false) . 'maintenanceUser/viewItem/'. $this->getArgument(0).  $linkSuffix,
                'label'     => $this->getCrudConfig(self::CONFIG_BASE_LANGUAGE_TAG) . '.menu.back.to.maintenanceuser',
                'img'       => '/resources/images/icons/arrow-180.png',
                'requires'  => $this->getCrudConfig(self::CONFIG_PERMISSION_NEW)
            );
        }

        // add addition menu items
        $menuItemsConfig = array_merge($menuItemsConfig, $this->getAdditionalOverviewMenuItems());

        ACL_PermissionManager::filterAccessAllowedOnConfiguration($menuItemsConfig);

        // setup link for overview item
        $overviewLinkTemplate = $this->getBaseRequest() . 'viewItem/%1$d' . $linkSuffix;

        $overviewForm = $this->getOverviewForm($overviewLinkTemplate);
        $overviewForm->filterOverViewForMaintenanceUserId($this->getArgument(0));

        // set label for the form
        if (!$overviewForm->getLabel()) {
            $overviewLabel = __($this->getCrudConfig(self::CONFIG_BASE_LANGUAGE_TAG) . '.overview.title', $overviewForm->getObjectsCount());
            $overviewForm->setLabel($overviewLabel);
        }

        $templateConfig = $this->getTemplateConfig();

        // a json request was done -> we will only out the form data
        if ($this->isJsonRequest()) {
            $overviewForm->setDisplayedViaAjax(true);
            $overviewForm->setDisplayDataOnly(true);
            foreach ($overviewForm->getControls() as $control) { /* @var $control Form_Controls_ControlAbstract */
                print $control->viewInit(false);
                print $control->view(false);
            }
            die();
        }


        pp($this->_displayDisplayOverviewAction($menuItemsConfig, $overviewForm, $templateConfig));
    }

    /**
     * Outputs the given file to the browser
     */
    public function displayMaintenanceImage()
    {
        // check for a vfs file id to be set in the url
        $vfsFileId = (isset($_GET['file'])) ? intval($_GET['file']) : null;

        if ($vfsFileId !== null) {

            try {
                // try to fetch the vfs file from the file system
                $vfsFile = VFS_File::getById($vfsFileId);

                // create the file system with the files namespace
                $fileSystem = new Filesystem($vfsFile->getVfsNamespace()->getNamespace());

                // output the file and die
                $fileSystem->outputFile($vfsFileId);
            }
            catch (LoadException $exception) {
                // set header to 404
                header("HTTP/1.1 404 Not Found");
            }
        }
        else {
            // file is not found set 404 not found
            header("HTTP/1.1 404 Not Found");
        }

        // stop execution, file is outputted or not found
        die();
    }

    /**
     * Retrieve overview form
     *
     * @param string $linkTemplate (Optional)
     * @return FormOverviewAbstract
     */
    private function getOverviewForm($linkTemplate = null)
    {
        if ($this->getCrudConfig(self::CONFIG_FORM_NAME_OVERVIEW) === null) {
            throw new InvalidArgumentException(sprintf('%1$s: Config not set for overview form', __METHOD__));
        }

        $itemsMethod        = null;
        $itemsCountMethod   = null;

        if ($this->getCrudConfig(self::CONFIG_ITEMS_METHOD) !== null && $this->getCrudConfig(self::CONFIG_ITEMS_COUNT_METHOD) !== null) {
            $dataSource         = $this->getDataSource();
            $itemsMethod        = array($dataSource, $this->getCrudConfig(self::CONFIG_ITEMS_METHOD));
```

```
            $itemsCountMethod   = array($dataSource, $this->getCrudConfig(self::CONFIG_ITEMS_COUNT_METHOD));
        }

        $reflectionRelationClass = new ReflectionClass($this->getCrudConfig(self::CONFIG_FORM_NAME_OVERVIEW));
        return $reflectionRelationClass->newInstanceArgs(array(
            $itemsMethod,
            $itemsCountMethod,
            $linkTemplate
        ));
    }

    /**
     * Method starts the testing of the mobile application API for the OA iOS app
     */
    public function execTest()
    {
        // log the start of the test
        $currentStartTime = time();
        echo sprintf('------ Starting tests --------<br/>Timestamp: %1$s.<br>', $currentStartTime);
        echo sprintf('Failed tests will be displayed here...<br/><br/>');

        // execute each test to identify failures in API method calls
        /*$this->testAuthenticationRequestAPIMethod($testAPIURL);
        $this->testSynchronizationAPIMethod($testAPIURL);
        $this->testAddMaintenanceAPIMethod($testAPIURL);
        $this->testAddressLookupAPIMethod($testAPIURL);
        $this->testAddUserAPIMethod($testAPIURL);
        $this->testAllCustomerAPIMethod($testAPIURL);
        $this->testAvailabilityDatesAPIMethod($testAPIURL);
        $this->testCancelReservationAPIMethod($testAPIURL);*/
        $this->testCheckStateAPIMethod();
        $this->testImageAPIMethod($testAPIURL);

        // log the end of the test
        echo sprintf('<br/>------ Finished Tests in %1$s seconds. -------', time() - $currentStartTime);
    }

    /**
     * Returns whether the status is set in the json object
     *
     * @param resource $connection
     * @param string $status
     * @param stdObject $JSONObject
     * @return bool
     */
    private function validateCurlConnectionForStatus($connection = null, $status, $debugInfo)
    {
        if ($connection !== null) {

            // execute curl and release the connection
            $JSONResult = json_decode(curl_exec($connection));
            curl_close($connection);

            // check the result
            if (!isset($JSONResult->{'status'}) || $JSONResult->{'status'} != $status) {
                echo sprintf('Test failed: %1$s failed with received status: "%2$s".<br/>', $debugInfo, $JSONResult->{'status'});
            }
        }
    }

    /**
     * Create and configures the curl resource
     *
     * @param string $urlName
     * @param null $postArray
     * @param bool $authenticate
     * @return resource
     */
    private function createCurlConnectionForUrlWithPostArrayAndAuthentication($urlName = '', $postArray = null, $authenticate = true)
    {
        // create the curl connection
        $url = 'http://stephan.bc-dev.dev.solware.local/api/v1/';
        $currentCurlObject = curl_init(sprintf('%1$s%2$s', $url, $urlName));
        curl_setopt($currentCurlObject, CURLOPT_RETURNTRANSFER, true);

        // when authentication should be provided add auth header
        if ($authenticate === true) {
            curl_setopt($currentCurlObject, CURLOPT_HTTPHEADER, array(
                sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
            ));
        }

        // when post value is set set as post
        if ($postArray !== null) {
            curl_setopt($currentCurlObject, CURLOPT_POST, true);
            curl_setopt($currentCurlObject, CURLOPT_POSTFIELDS, $postArray);
        }

        // return the curl resource
        return $currentCurlObject;
    }

    /**
     * Test whether image delivery is working OK
     *
     * @param string $URL
     */
    private function testImageAPIMethod($URL)
    {
        // create the connection
        $connection = $this->createCurlConnectionForUrlWithPostArrayAndAuthentication('getImage/', array(), true);
        curl_setopt($connection, CURLOPT_HEADER, false);

        // execute connection
        $result = curl_exec($connection);
        curl_close($connection);

        if (strpos($result, 'application/json') == 0) {
            echo sprintf('Test failed: Image API fetched with wrong headers.');
        }

        /*  Test a valid request where the unauthorized is trying to append a reservation    */
        $currentCURLObject = curl_init(sprintf('%1$sgetImage/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_VERBOSE, true);
        curl_setopt($currentCURLObject, CURLOPT_HEADER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
```

```
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'type'      =>  'Location',
            'id'        =>  '2',
        ));

        $result = curl_exec($currentCURLObject);
        curl_close($currentCURLObject);

        // image is expected for this call
        if (strpos($result, 'image/jpeg') == 0) {
            echo sprintf('Test failed: Image API fetched with wrong headers.');
        }

        /*  Test a valid request where the unauthorized is trying to append a reservation    */
        $currentCURLObject = curl_init(sprintf('%1$sgetImage/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_VERBOSE, true);
        curl_setopt($currentCURLObject, CURLOPT_HEADER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'type'      =>  'Location',
            'id'        =>  '1',
        ));

        $result = curl_exec($currentCURLObject);
        curl_close($currentCURLObject);

        // image is expected for this call
        if (strpos($result, 'application/json') == 0) {
            echo sprintf('Test failed: Image API fetched with wrong headers.');
        }
    }

    /**
     * Test the check state API method with multiple calls
     *
     * @param string $URL
     */
    private function testCheckStateAPIMethod()
    {
        // create connection with invalid parameters
        $connection = $this->createCurlConnectionForUrlWithPostArrayAndAuthentication('checkState/', array(), true);
        $this->validateCurlConnectionForStatus($connection, 'error', 'Check state API method');

        // create test for valid connection
        $connection = $this->createCurlConnectionForUrlWithPostArrayAndAuthentication('checkState/', array(
            'reservation_list'  =>  '13,14,15, 1234'
        ), true);
        $this->validateCurlConnectionForStatus($connection, 'success', 'Check state API method');

        /*  Not tests are required more, previous test shows that unknown reservation do not get handled    */
    }

    /**
     * Tests the cancelling of a reservation method
     *
     * @param string $URL
     */
    private function testCancelReservationAPIMethod($URL)
    {
        /*  Test ainvalid request with no Parameters    */
        $currentCURLObject = curl_init(sprintf('%1$scancelReservation/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));

        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check the result
        $this->validateStatusForJSON('error', $JSONResult, 'Cancel Reservation API method');

        /*  Test a valid request where the authorized accoutn can actually cancel the reservation    */
        $currentCURLObject = curl_init(sprintf('%1$scancelReservation/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'reservation_id'    =>  13,
        ));

        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check the result
        $this->validateStatusForJSON('success', $JSONResult, 'Cancel Reservation API method');

        /*  Test a valid request where the unauthorized is trying to append a reservation    */
        $currentCURLObject = curl_init(sprintf('%1$scancelReservation/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'reservation_id'    =>  14,
        ));

        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check the result
        $this->validateStatusForJSON('error', $JSONResult, 'Cancel Reservation API method');

        /*  Test a valid request where an unknown reservation will be cancelled    */
        $currentCURLObject = curl_init(sprintf('%1$scancelReservation/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
```

```
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'reservation_id'    =>  12345,
        ));

        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check the result
        $this->validateStatusForJSON('error', $JSONResult, 'Cancel Reservation API method');
    }

    /**
     * Test whether the availability dates method is available
     *
     * @param string $URL
     */
    private function testAvailabilityDatesAPIMethod($URL)
    {
        /*  Test a valid response with no Parameters    */
        $currentCURLObject = curl_init(sprintf('%1$sgetAvailableDates/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));

        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check the result
        $this->validateStatusForJSON('success', $JSONResult, 'Availability Dates API method');
    }

    /**
     * Test the fetch of customers
     *
     * @param $URL
     */
    private function testAllCustomerAPIMethod($URL)
    {
        /*  Test a valid response with no Parameters    */
        $currentCURLObject = curl_init(sprintf('%1$sallCustomers/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));

        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check the result
        if (!isset($JSONResult->{'status'}) || $JSONResult->{'status'} != 'success') {
            echo sprintf('Test failed: Fetching customer method failed with received status: "%1$s".<br/>', $JSONResult->{'status'});
        }
    }

    /**
     * Test the adding of a new user or maintenance object method
     *
     * @param $URL
     */
    private function testAddUserAPIMethod($URL)
    {
        /*  Test a valid response with no Parameters    */
        $currentCURLObject = curl_init(sprintf('%1$saddUser/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));

        // execute the request
        $JSONResult =  json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        $this->validateStatusForJSON('error', $JSONResult, 'Add User API method');

        /*  Test a valid response with no an invalid customer    */
        $currentCURLObject = curl_init(sprintf('%1$saddUser/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'name'          =>  'Stephan de Bakker',
            'email'         =>  's.debakker@solware.tst',
            'phone_number'  =>  '+031(06)41143752',
            'zipcode'       =>  '2523GB',
            'house_number'  =>  '45',
            'customer_id'   =>  '50',
            'date'          =>  time(),
            'daypart_id'    =>  '2',
        ));

        // execute the request
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        $this->validateStatusForJSON('error', $JSONResult, 'Add User API method');

        /*  Location validation is already covered in its individual method, this uses the same method  */
        /*  Test a invalid request with no valid daypart    */
        $currentCURLObject = curl_init(sprintf('%1$saddUser/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'name'          =>  'Stephan de Bakker',
```

```
            'email'         =>  's.debakker@solware.tst',
            'phone_number'  =>  '+031(06)41143752',
            'zipcode'       =>  '2523GB',
            'house_number'  =>  '45',
            'customer_id'   =>  '50',
            'date'          =>  time(),
            'daypart_id'    =>  '18',
        ));

        // execute the request
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);
        $this->validateStatusForJSON('error', $JSONResult, 'Add User API method');

        /*
        **  Email validation is used by the SEF framework that is around for a couple of years and already has been tested, telephone
        **  is identical to the validation method that is used on the mobile phone. When the request is executed from the mobile application
        **  no unknown errors should occur. When otherwise (some kind of hacked) the request is executed it doesnt matter just make sure no
        **  SQL injection or other dangerous values. SQL injection prevention is standard in POST handling of the Framework. Dat validation will
        **  occur in an individual method and will be tested outside this method
        */
    }

    /**
     * Test all possible API calls and different ways of calling those API methods
     */
    private function testAuthenticationRequestAPIMethod($URL)
    {
        // first test a default authentication request, will be tested withou a UUID, with a correct formatted UUID and with misformatted UUID's
        $currentCURLObject = curl_init(sprintf('%1$sauthenticationRequest', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);

        // execute and decode the result
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // when the status is not set the test failed
        if (!isset($JSONResult->{'status'})) {
            echo sprintf('Test failed: Authentication Request failed, no status result is set in the response for empty auth request.<br/>');
        } else if ($JSONResult->{'status'} != 'fail'){
            echo sprintf('Test failed: Authentication Request failed, expected result for an empty request is fail but result = "%1$s"<br/>',
$JSONResult['status']);
        }

        // next test case provides a valid UUID
        $currentCURLObject = curl_init(sprintf('%1$sauthenticationRequest', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array('UUID' => 'B2A4993A-E131-4B69-8917-9FD49AF78168'));

        // execute and decode the result
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // the status should be set to failed
        if (!isset($JSONResult->{'status'})) {
            echo sprintf('Test failed: Authentication request with a valid UUID has failed because no status was set.<br/>');
        } else if ($JSONResult->{'status'} != 'success') {
            echo sprintf('Test failed: Authentication request failed with valid UUID format "B2A4993A-E131-4B69-8917-9FD49AF78168" and result:
"%1$s"<br/>', $JSONResult->{'status'});
        }

        // next test case provides an invalid UUID
        $currentCURLObject = curl_init(sprintf('%1$sauthenticationRequest', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array('UUID' => 'B2A4993A-E131-4B69-8917-9FD49AF7816')); /* Invalid UUID    */

        // execute and decode result
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // the status should be set to failed
        if (!isset($JSONResult->{'status'})) {
            echo sprintf('Test failed: Authentication request with an invalid UUID has failed because no status was set.<br/>');
        } else if ($JSONResult->{'status'} != 'fail') {
            echo sprintf('Test failed: Authentication request failed with invalid UUID format "B2A4993A-E131-4B69-8917-9FD49AF7816" and result:
"%1$s"<br/>', $JSONResult->{'status'});
        }

        // next test case provides an invalid UUID
        $currentCURLObject = curl_init(sprintf('%1$sauthenticationRequest', $URL));
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array('UUID' => 'B2A4993A-E131-4B69-8917-9FD49AF781654')); /* Invalid UUID    */

        // execute and decode result
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // the status should be set to failed
        if (!isset($JSONResult->{'status'})) {
            echo sprintf('Test failed: Authentication request with an invalid UUID has failed because no status was set.<br/>');
        } else if ($JSONResult->{'status'} != 'fail') {
            echo sprintf('Test failed: Authentication request failed with invalid UUID format "B2A4993A-E131-4B69-8917-9FD49AF7816" and result:
"%1$s"<br/>', $JSONResult->{'status'});
        }
    }

    /**
     * Executes the test for synchronization API method
     */
    private function testSynchronizationAPIMethod($URL)
    {
        // create request
        $currentCURLObject = curl_init(sprintf('%1$ssynchronizeDefects/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));

        // execute and decode the result
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check whether the status is success
        if (!isset($JSONResult->{'status'})) {
            echo sprintf('Test failed: Synchronize defects method failed not status was set by the API.<br/>');
```

```
        } else if ($JSONResult->{'status'} != 'success') {
            echo sprintf('Test failed: Synchronize defects method failed because of unexpected status: "%1$s".<br/>', $JSONResult->{'status'});
        }
    }

    /**
     * Tests the address lookup method
     */
    private function testAddressLookupAPIMethod($URL)
    {
        // create test connection with no parameters in POST
        $currentCURLObject = curl_init(sprintf('%1$slookUpAddress/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));

        // execute the request
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check the status
        if (!isset($JSONResult->{'status'}) || $JSONResult->{'status'} != 'error') {
            echo sprintf('Test failed: Address lookup with no parameters returned invalid status, received: "%1$s".<br/>', $JSONResult->{'status'});
        }

        /*  Test with a valid zipcode and housenumber   */
        $currentCURLObject = curl_init(sprintf('%1$slookUpAddress/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'zipcode'       =>  '2523GB',
            'house_number'  =>  '45',
        ));

        // execute the request
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check result
        if (!isset($JSONResult->{'status'}) || $JSONResult->{'status'} != 'success') {
            echo sprintf('Test failed: address lookup with valid post return wrong result, received: "%1$s".<br/>', $JSONResult->{'status'});
        }

        /*  Test with a valid zipcode and housenumber but with a space between   */
        $currentCURLObject = curl_init(sprintf('%1$slookUpAddress/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'zipcode'       =>  '2523 GB',
            'house_number'  =>  '45',
        ));

        // execute the request
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check result
        if (!isset($JSONResult->{'status'}) || $JSONResult->{'status'} != 'success') {
            echo sprintf('Test failed: address lookup with valid post return wrong result, received: "%1$s".<br/>', $JSONResult->{'status'});
        }

        /*  Test with invalid zipcode housenumber combination   */
        $currentCURLObject = curl_init(sprintf('%1$slookUpAddress/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'zipcode'       =>  '1234AB',
            'house_number'  =>  '999',
        ));

        // execute the request
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check result
        if (!isset($JSONResult->{'status'}) || $JSONResult->{'status'} != 'error') {
            echo sprintf('Test failed: address lookup with invalid post return wrong result, received: "%1$s".<br/>', $JSONResult->{'status'});
        }

        /*  The addition does not need to be tested because this addition field is not validated with the database  */
    }

    /**
     * Test the adding of a maintenance method
     */
    private function testAddMaintenanceAPIMethod($URL)
    {
        // create test connection with no parameters in POST
        $currentCURLObject = curl_init(sprintf('%1$saddMaintenance/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));

        // execute request and decode
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // the result must be failed
        if (!isset($JSONResult->{'status'}) || $JSONResult->{'status'} != 'error') {
            echo sprintf('Test failed: Add maintenance failed with no POST arguments, wrong status given: "%1$s"<br/>', $JSONResult->{'status'});
        }
```

```
        /*  Now send a request with all the correct parameters, also the user has access to the provided object  */
        $currentCURLObject = curl_init(sprintf('%1$saddMaintenance/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'defect_id'             =>  1,
            'user_id'               =>  13,
            'is_final_maintenance'  =>  'false',
        ));

        // execute request and decode
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check result
        if (!isset($JSONResult->{'status'}) || $JSONResult->{'status'} != 'success') {
            echo sprintf('Test failed: Add maintenance with correct parameters failed, received status: "%1$s".<br/>', $JSONResult->{'status'});
        }

        /*  NExt test case includes a wrong defect_id but all other values are correct  */
        $currentCURLObject = curl_init(sprintf('%1$saddMaintenance/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'defect_id'             =>  1234,
            'user_id'               =>  13,
            'is_final_maintenance'  =>  'false',
        ));

        // execute request and decode
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check correct result
        if (!isset($JSONResult->{'status'}) || $JSONResult->{'status'} != 'error') {
            echo sprintf('Test failed: Add maintenance with invalid defect id did not return correct status, received "%1$s".<br/>', $JSONResult-
>{'status'});
        }

        /*  Test a request that a person with the authorization has no access to the object */
        $currentCURLObject = curl_init(sprintf('%1$saddMaintenance/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'defect_id'             =>  1,
            'user_id'               =>  14,
            'is_final_maintenance'  =>  'false',
        ));

        // execute request and decode
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // the result should be error
        if (!isset($JSONResult->{'status'}) || $JSONResult->{'status'} != 'error') {
            echo sprintf('Test failed: Add maintenance method user that does not have access failed because it was authenticate to add a new
maintenance, received status: "%1$s".<br/>', $JSONResult->{'status'});
        }

        /*  Final test for adding a maintenance checks whether the user actually exists */
        $currentCURLObject = curl_init(sprintf('%1$saddMaintenance/', $URL));
        curl_setopt($currentCURLObject, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($currentCURLObject, CURLOPT_POST, true);
        curl_setopt($currentCURLObject, CURLOPT_HTTPHEADER, array(
            sprintf('Authorization: Basic %1$s',
base64_encode('XkHTs3UWz50ykkBJgHWfXFzjogD3OZR6TJqEg2RYLrVuqJejt0rqnyX1sU7FpSQS:OWnLTznvhDJvACxjuKS7R80ZIQ0Tn6ubfk2PDKIkeNoIk0lJaavVOQYeICZnWrwP')),
        ));
        curl_setopt($currentCURLObject, CURLOPT_POSTFIELDS, array(
            'defect_id'             =>  1,
            'user_id'               =>  3975,
            'is_final_maintenance'  =>  'false',
        ));

        // execute request and decode
        $JSONResult = json_decode(curl_exec($currentCURLObject));
        curl_close($currentCURLObject);

        // check the result
        if (!isset($JSONResult->{'status'}) || $JSONResult->{'status'} != 'error') {
            echo sprintf('Test failed: Add maintenance failed with invalid result, a user does not exist but is probably added, received: "%1$s".<br/
>', $JSONResult->{'status'});
        }
    }
}
```

```php
<?php

/**
 * OM_MaintenanceUserController - The user who has sent the maintenance, contact and adress info.
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */
class OM_MaintenanceUserController extends OM_MaintenanceUserControllerBase
{

    /**
     * Crud config, disable edit / new /remove
     * @var array
     */
    protected $_config = array(
        self::CONFIG_ENTITY_NAME            => OM_MaintenanceUser::TABLE,
        self::CONFIG_BASE_LANGUAGE_TAG      => 'om.maintenance.user',
        self::CONFIG_FORM_NAME_EXTENDED     => 'OM_MaintenanceUserForm',
        self::CONFIG_FORM_NAME_OVERVIEW     => 'OM_MaintenanceUserOverviewForm',
        self::CONFIG_PERMISSION_EDIT        => OM_MaintenanceUser::PERMISSION_EDIT,
        self::CONFIG_PERMISSION_NEW         => OM_MaintenanceUser::PERMISSION_NEW,
        self::CONFIG_PERMISSION_REMOVE      => OM_MaintenanceUser::PERMISSION_REMOVE,
        self::CONFIG_PERMISSION_VIEW        => OM_MaintenanceUser::PERMISSION_VIEW,

        parent::CONFIG_EXTENDED_TAB_COLOR   => 'green',
        parent::CONFIG_EXTENDED_TABS        => array()
    );

    /**
     * This method marks the given maintenance user as spam and will automatically set any future reservations of this user as spam
     * which are not show in the overview of maintenance users.
     */
    public function markAsSpam()
    {
        try {
            // get the maintenance user that has to be marked as spam
            $omMaintenanceUser = OM_MaintenanceUser::getById($this->getArgument(0, null));

            // TODO: perform action to mark this user and following reservations from this user as spam and archive immediatly
        } catch (LoadException $exception) {

        }

        // return to the maintenance overview
        $this->redirect(sprintf('%1$sdisplayOverview/', $this->getBaseRequest()));
    }

    /**
     * Marks the current maintenance as archived and removes it from the default overview form
     */
    public function setMaintenanceArchived()
    {
        try {
            // load the maintenance that will be archived
            $omMaintenanceUser = OM_MaintenanceUser::getById($this->getArgument(0, null));

            // set that this maintenance is archived
            $omMaintenanceUser->setIsArchived(true);
            $omMaintenanceUser->save();
        } catch (LoadException $exception) {
            throw $exception;
        }

        // return to the form page of the current maintenance
        $this->redirect(sprintf('%1$sdisplayOverview/', $this->getBaseRequest()));
    }

    /**
     * This method marks the given maintenance user with ID as processed in MER and immeditatly returns to the overview, in the overview
     * this object will have a marker defining that it has been processed already an kan be set as accepted state
     */
    public function markAsProcessed()
    {
        try {
            // fetch the associated maintenance user
            $omMaintenanceUser = OM_MaintenanceUser::getById($this->getArgument(0, null));

            // set as processed in MER and save this user
            $omMaintenanceUser->setIsProcessedInMer(true);
            $omMaintenanceUser->save();
        } catch (LoadException $exception) {
            // rethrow exception
            throw $exception;
        }

        // always return to the overview, also when nothing has been changed then the maintenance user does not exist. This means the URL is invalid
        $this->redirect(sprintf('%1$sdisplayOverview/', $this->getBaseRequest()));
    }

    /**
     * Shows a new form with a textarea to fill with the reason of denial, when submitted the user will be sent a new email message
     * with information why the reservation has been denied by BouwMeester.
     *
     * When the user does not exist it returns to the overview of maintenance users.
     */
    public function changeStateDenied()
    {
        try {
            // get the maintance user object to deny
            $omMaintenanceUser = OM_MaintenanceUser::getById($this->getArgument(0, null));

            // create the form to send a new email to the user with the provided denial reason
            $denialForm = new OM_MaintenanceUserDenialForm();

            // when the form is posted and valid start sending the email to the user
            if ($denialForm->isPosted() && $denialForm->isValid()) {

                // retrieve the object to send and set the denial reason and state to denied, also archive the maintenance
                $omMaintenanceUser->setDenialReason($denialForm->getControlByName(OM_MaintenanceUserDenialForm::FORM_DENIAL_REASON)->getValue());
                $omMaintenanceUser->setOmMaintenanceStateId(OM_MaintenanceState::DENIED_STATE);
                $omMaintenanceUser->setIsArchived(true);

                // create and send the message
                $denialMessage = new OM_MaintenanceUserDeniedMessage($omMaintenanceUser);
                $denialMessage->send();

                // create the history object to log that the denied state has been selected an the uses has been mailed with state info
                $omMaintenanceUserStateHistory = new OM_MaintenanceUserStateHistory();

                // configure the history object
```

```
                $omMaintenanceUserStateHistory->setOmMaintenanceStateId(OM_MaintenanceState::DENIED_STATE);
                $omMaintenanceUserStateHistory->setOmMaintenanceUser($omMaintenanceUser);
                $omMaintenanceUserStateHistory->setStateCorrespondingEmailWasSent(true);
                $omMaintenanceUserStateHistory->setStateChangedDate(Sef_Helpers_DateTime::getSqlDateTimeByTimestamp(time()));

                // save the reason to the maintenance user object
                $omMaintenanceUser->save();
                $omMaintenanceUserStateHistory->save();

                // set success message to notify the user that the message is successfully sent
                $this->getView()->setFlashMessageDirect(sprintf(__('om.maintenanceuser.deny.action.success'), $this-
>getBaseRequest() .'displayOverview/'), Sef_Routing_Mvc_View::FLASHMESSAGE_TYPE_SUCCESS);
            }

            // get the menu items
            $menuItemsConfig = array(
                array(
                    'href'     => $this->getBaseRequest(). 'viewItem/'. $omMaintenanceUser->getOmMaintenanceUserId(),
                    'label'    => 'core.back',
                    'img'      => '/resources/images/icons/arrow-180.png',
                    'requires' => OM_MaintenanceUser::PERMISSION_VIEW
                ),
            );

            // show the form
            return Sef_Helpers_Layout::displayDefaultTemplate($menuItemsConfig, null, $denialForm);
        } catch (LoadException $exception) {
            // the maintenance user id does not exist, just return to the overview of maintenances
            $this->redirect(sprintf('%1$sdisplayOverview/', $this->getBaseRequest()));
        }
    }

    /**
     *  Returns the additional view items for the individual view of a maintenance user
     *
     * @param DataObject $entityObject
     * @return array
     */
    protected function getAdditionalViewItemMenuItems(DataObject $entityObject)
    {
        $menuItems = array();

        if ($entityObject instanceof OM_MaintenanceUser) {

            // when the maintenance user is not yet processed in MER show this option to do so
            if ($entityObject->getIsProcessedInMer() == false) {
                $menuItems[] = array(
                    'href'      =>  $this->getBaseRequest() . 'markAsProcessed/'. $entityObject->getOmMaintenanceUserId(). '/?'. $this-
>getHistoryData()->generateCurrentLinkSuffix(),
                    'label'     =>  'om.maintenanceuser.view.mark.processed.title',
                    'img'       =>  '/resources/images/icons/marker--plus.png',
                    'requires'  =>  OM_MaintenanceUser::PERMISSION_EDIT,
                );
            }

            // when the maintenance object is not yet already on the state of denial show the menu item
            if ($entityObject->getOmMaintenanceStateId() != OM_MaintenanceState::DENIED_STATE) {
                $menuItems[] = array(
                    'href'      =>  $this->getBaseRequest(). 'changeStateDenied/'. $entityObject->getOmMaintenanceUserId(). '/?'. $this-
>getHistoryData()->generateCurrentLinkSuffix(),
                    'label'     =>  'om.maintenanceuser.view.mark.state.denied',
                    'img'       =>  '/resources/images/icons/box--exclamation.png',
                    'requires'  =>  OM_MaintenanceUser::PERMISSION_EDIT,
                );
            }

            // show the archived button when not yet archived
            if ($entityObject->getIsArchived() != true) {
                $menuItems[] = array(
                    'href'      =>  $this->getBaseRequest(). 'setMaintenanceArchived/'. $entityObject->getOmMaintenanceUserId(). '/?'. $this-
>getHistoryData()->generateCurrentLinkSuffix(),
                    'label'     =>  'om.maintenanceuser.view.mark.archived.title',
                    'img'       =>  '/resources/images/icons/vise-drawer.png',
                    'requires'  =>  OM_MaintenanceUser::PERMISSION_EDIT,
                );
            }

            // add the report as spam button
            /*$menuItems[] =
                array(
                    'href'      => $this->getBaseRequest() . 'reportAsSpam/' . $entityObject->getOmMaintenanceUserId(). '/?' . $this-
>getHistoryData()->generateCurrentLinkSuffix(),
                    'label'     => 'om.maintenanceuser.view.report.title',
                    'img'       => '/resources/images/icons/auction-hammer--minus.png',
                    'requires'  => OM_MaintenanceUser::PERMISSION_EDIT,
                    'onclick'   => 'return confirm(\'' . __('om.maintenanceuser.view.report.confirm.description') . '\')',
                );
            */
        }

        return $menuItems;
    }

    /**
     * (non-PHPdoc)
     * @see Sef_Routing_Mvc_ControllerCruds::_displayViewItemAction()
     */
    protected function _displayViewItemAction(DataObject $entityObject, array $menuItemsConfig = null, array $tabItemsConfig = null, $forms = null,
array $extraOptions = null)
    {
        // when the object is a OM_MaintenanceUser create the added forms
        if ($entityObject instanceof OM_MaintenanceUser) {

            // create the overview form for defects coupled to this maintenance user
            $omMaintenanceMaintenanceUserOverviewForm = new OM_MaintenanceUserMaintenanceOverviewForm(array($entityObject,
'getAllRelatedOmMaintenanceThruOmMaintenanceUserId'), array($entityObject, 'getCountAllRelatedOmMaintenanceThruOmMaintenanceUserId'));
            $omMaintenanceMaintenanceUserOverviewForm->setLabel(__('om.maintenanceuser.viewitem.all.maintenance.title'));

            // create the overview form for state history changes with email information
            $omMaintenanceUserStateHistoryMaintenanceUserOverviewForm = new
OM_MaintenanceUserMaintenanceUserStateHistoryOverviewForm(array($entityObject, 'getAllRelatedOmMaintenanceuserstatehistoryThruOmMaintenanceUserId'),
array($entityObject, 'getCountAllRelatedOmMaintenanceuserstatehistoryThruOmMaintenanceUserId'));
            $omMaintenanceUserStateHistoryMaintenanceUserOverviewForm->setLabel(__('om.maintenanceuser.viewitem.all.state.history.title'));

            if (is_array($forms)) {
                $forms[] = $omMaintenanceMaintenanceUserOverviewForm;
                $forms[] = $omMaintenanceUserStateHistoryMaintenanceUserOverviewForm;
            } else {
                $forms = array($forms, $omMaintenanceMaintenanceUserOverviewForm, $omMaintenanceUserStateHistoryMaintenanceUserOverviewForm);
            }
```

```
        }

        parent::_displayViewItemAction($entityObject, $menuItemsConfig, $tabItemsConfig, $forms, $extraOptions);
    }

    /**
     * Display the result of the displayOverview action
     * Shows error message when needed with all the failed approved objects and the succeeded approved objects
     *
     * @param array $menuItemsConfig
     * @param array|Form $form
     * @param array $templateConfig
     */
    protected function _displayDisplayOverviewAction(array $menuItemsConfig = null, $form, $templateConfig)
    {
        // when the form is successfully posted in overview form set the message that all messages have been sent
        if ($form->isPosted()) {

            if ($form->save()) {
                // set success message to notify the user that the message is successfully sent
                $this->getView()->setFlashMessageDirect(sprintf(__('om.maintenanceuser.approve.action.success'), $form->getTotalProcessed()),
Sef_Routing_Mvc_View::FLASHMESSAGE_TYPE_SUCCESS);
            } else {
                $errors = '<ul>';

                // add the first message to include the amount of processed reservation
                if ($form->getTotalProcessed() > count($form->getProcessingErrors())) {
                    $errors .= sprintf(__('om.maintenanceuser.approve.action.success'), $form->getTotalProcessed());
                }

                // generate the error message
                foreach ($form->getProcessingErrors() as $errorString) {
                    $errors .= sprintf('<li>%1$s</li>', $errorString);
                }

                $errors .= '</ul>';
                $this->getView()->setFlashMessageDirect($errors, Sef_Routing_Mvc_View::FLASHMESSAGE_TYPE_INFO);
            }
        }

        // return the template form
        return Sef_Helpers_Layout::displayDefaultTemplate($menuItemsConfig, null, $form, $templateConfig);
    }
}
```

```php
<?php

/**
 * OM_MaintenanceUserForm
 *
 * @copyright   Copyright (c) 2007-2013 Solware B.V. (http://www.solware.nl)
 */
class OM_MaintenanceUserForm extends OM_MaintenanceUserFormBase
{
    /**
     * Override this method to make sure a custom device token can not be selected when editing
     *
     * @return array|null
     */
    protected function getApnsDeviceIdFormConfig()
    {
        // when editing do not show the apns device id field
        if ($this->getDisplayMode() != self::DISPLAY_MODE_EDIT) {
            return parent::getApnsDeviceIdFormConfig();
        } else {
            // return a null array
            return null;
        }
    }

    /**
     * (non-PHPdoc)
     * @see FormExtendedAbstract::_getGroupedFieldsColumns()
     */
    protected function _getGroupedFieldsColumns()
    {

        // the first group is always visible
        $groups = array();
        $groups[] = array(
            self::GROUP_LABEL   => '',
            self::GROUP_COLUMN  => self::GROUP_COLUMN_LEFT,
            self::GROUP_ITEMS   => array(
                OM_MaintenanceUserBase::RM_CUSTOMER_ID,
                OM_MaintenanceUserBase::NAME,
                OM_MaintenanceUserBase::EMAIL,
                OM_MaintenanceUserBase::OM_MAINTENANCE_STATE_ID,
                OM_MaintenanceUserBase::DENIAL_REASON,
                OM_MaintenanceUserBase::IS_ARCHIVED,
                OM_MaintenanceUserBase::IS_PROCESSED_IN_MER,
                ),
        );

        $groups[] = array(
            self::GROUP_LABEL       =>  __('om.maintenanceuser.address.column.title'),
            self::GROUP_COLUMN      =>  self::GROUP_COLUMN_RIGHT,
            self::GROUP_ITEMS       =>  array(
                OM_MaintenanceUser::ADRESS,
                OM_MaintenanceUser::HOUSE_NUMBER,
                OM_MaintenanceUser::ADRESS_ADDITION,
                OM_MaintenanceUser::CITY,
                OM_MaintenanceUser::ZIPCODE,
                OM_MaintenanceUserBase::PHONE_NUMBER_1,
                OM_MaintenanceUserBase::PHONE_NUMBER_2,
            ),
        );

        // return the groups to be shown
        return $groups;
    }
}
```

```php
<?php
/**
 * OM_MaintenanceUserRelatedMaintenanceForm
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */

class OM_MaintenanceUserMaintenanceOverviewForm extends OM_MaintenanceOverviewFormBase
{
    /**
     * Returns the columns which can be sorted in this form
     *
     * @return array
     */
    protected function getSortColumns()
    {
        return array();
    }

    /**
     * Retrieve fields for columns
     *
     * @return array
     */
    protected function getColumnsOverviewForm()
    {
        return array(
            OM_Maintenance::OM_DEFECT_ID,
            OM_Maintenance::VFS_FILE_ID,
            OM_Maintenance::DESCRIPTION,
        );
    }

    /**
     * Return the file link in the form
     *
     * @return array
     */
    public function getVfsFileIdColumnOverviewFormConfig()
    {
        return array(
            'sortField' =>  null,
            'label'     => 'om.maintenance.file.title',
            'width'     => '*',
            'property'  => 'getDisplayOverviewMaintenanceFile',
        );
    }
}
```

```php
<?php
/**
 * OM_MaintenanceUserMaintenanceUserStateHistoryOverviewForm
 *
 * @copyright   Copyright (c) 2007-2011 Solware B.V. (http://www.solware.nl)
 */

class OM_MaintenanceUserMaintenanceUserStateHistoryOverviewForm extends OM_MaintenanceUserStateHistoryOverviewForm
{
    /**
     * Returns the columns which can be sorted in this form
     *
     * @return array
     */
    protected function getSortColumns()
    {
        return array();
    }

    /**
     * Retrieve fields for columns
     *
     * @return array
     */
    protected function getColumnsOverviewForm()
    {
        return array(
            OM_MaintenanceUserStateHistory::OM_MAINTENANCE_STATE_ID,
            OM_MaintenanceUserStateHistory::STATE_CHANGED_DATE,
            OM_MaintenanceUserStateHistory::STATE_CORRESPONDING_EMAIL_WAS_SENT,
        );
    }

    /**
     * Returns the custom configuration for the date field in the state changed overview
     *
     * @return array
     */
    protected function getStateChangedDateColumnOverviewFormConfig()
    {
        return array(
            'sortField' =>  null,
            'label'     => 'om.maintenancestate.history.date.title',
            'width'     => '*',
            'property'  => 'getDisplayStateChangedDateOverview',
        );
    }
}
```