

CONVERTIGO SDK

THE ULTIMATE CLIENT MOBILE API FOR CONVERTIGO MBAAS



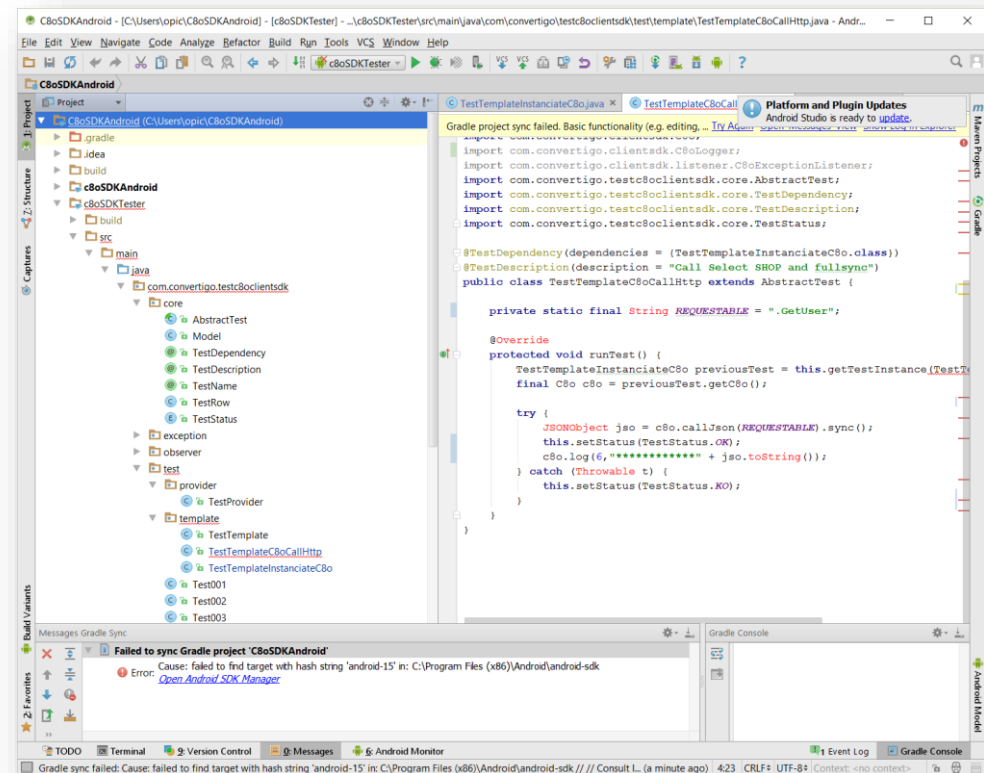
WHY CONVERTIGO SDK ?

- Abstracts Mobile app developer from protocol complexity
- Gives simple cross-platform API to access Convertigo MBaaS services
- Brings additional out of the box functionalities
 - Local Cache
 - Enhanced Communication Cryptography
 - Session management
 - Automatic UI Thread management
 - Mobile activity traced on Server.
- Error management
- Optional FullSync off line data managment



ABSTRACT THE DEVELOPPER FROM PROTOCOL COMPLEXITY

- Mobile applications usually accesses services though REST/JSON protocols.
- This access code must be written for each Mobile app, causing technical debt , preventing code sharing and useless coding time.
- Convertigo SDK simplifies all that by a providing a very simple API:
 - `C8O.CallJSON("project.service", "key", "value");`
 - All HTTP(S)/REST/JSON is done automatically
 - JSON objects are automatically created
 - Variables are automatically passed as key/values to sequences variables



DATA AUTOMATICALLY RETURNED AS JSON OBJECTS

- No use to parse JSON yourself !
- Data will be returned automatically as a JSON object you can use with your Java, Objective-C, Swift or C# code.
- Calls will be done automatically in asynchronous mode to prevent blocking UI Threads
- Callbacks will be called when data is ready so you can update your UI.

```
- (void)displayErrorMessage:(NSError *)error {
    NSString *titleString = [error localizedDescription];
    NSString *messageString = [error localizedFailureReason];
    NSString *moreString = [error localizedRecoverySuggestion] ? [error localizedRecoverySuggestion] :
        NSLocalizedString(@"Try again.", nil);

    messageString = [NSString stringWithFormat:@"%s", messageString, moreString];

    UIAlertController *alertView = [[UIAlertController alloc] initWithTitle:titleString
        message:messageString delegate:self
        cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alertView show];
}

- (IBAction)invokeCbo:(id)sender {
    // Hide keyboard of textFields on start.
    [_scrollView.cboTors resignFirstResponder];
    [_scrollView.param1 resignFirstResponder];
    [_scrollView.param2 resignFirstResponder];
    [_scrollView.param3 resignFirstResponder];
    [_scrollView.param4 resignFirstResponder];
    [_scrollView.param5 resignFirstResponder];
    [_scrollView.param6 resignFirstResponder];
    [_scrollView.value1 resignFirstResponder];
    [_scrollView.value2 resignFirstResponder];
    [_scrollView.value3 resignFirstResponder];
    [_scrollView.value4 resignFirstResponder];
    [_scrollView.value5 resignFirstResponder];
    [_scrollView.value6 resignFirstResponder];

    NSString *requestable = [NSString stringWithFormat:@"%s", _scrollView.cboTors.text];
    NSMutableDictionary *parameters = [NSMutableDictionary dictionary];

    if ([_scrollView.param1.text.length > 0] && [_scrollView.value1.text.length > 0])
        [parameters setObject:_scrollView.param1.text forKey:_scrollView.param1.text];
    if ([_scrollView.param2.text.length > 0] && [_scrollView.value2.text.length > 0])
        [parameters setObject:_scrollView.param2.text forKey:_scrollView.param2.text];
    if ([_scrollView.param3.text.length > 0] && [_scrollView.value3.text.length > 0])
        [parameters setObject:_scrollView.param3.text forKey:_scrollView.param3.text];
    if ([_scrollView.param4.text.length > 0] && [_scrollView.value4.text.length > 0])
        [parameters setObject:_scrollView.param4.text forKey:_scrollView.param4.text];
    if ([_scrollView.param5.text.length > 0] && [_scrollView.value5.text.length > 0])
        [parameters setObject:_scrollView.param5.text forKey:_scrollView.param5.text];
    if ([_scrollView.param6.text.length > 0] && [_scrollView.value6.text.length > 0])
        [parameters setObject:_scrollView.param6.text forKey:_scrollView.param6.text];

    [self saveFieldsToFile];
    [_activityIndicator startAnimating];

    // Call requestable created with textfields' content and with parameters the test variable value from
    // the textfield.
    [_cbo callRequestable:requestable
        withParameters:parameters
        andResponseType:(_useJSON ? @"JSON" : @"XML")
        success:^(id result) {
            NSLog(@"%s", result);
            [_cboResponse setText:[NSString stringWithFormat:@"%s", result]];
            [_activityIndicator stopAnimating];
        }
        failure:^(NSError *error) {
            // Oops!
            NSLog(@"%s", error);
            [_cboResponse setText:error.userInfo[@"NSHTMLContent"]];
            [_activityIndicator stopAnimating];

            [self displayErrorMessage:error];
        }];
}

@end
```

LOCAL CACHE

- Automatic local cache management
- Stores local responses from the server in a local mobile database
- Cache policy and Time to live can be configured at each call
- If a call to the same data is done within the time to live, the data will be retrieved automatically from the local database even if network is not present.

- Calling Sample

```
c8o.callJSON("myproject.myservice",
    "__localCache", {
        "enabled":true,
        "policy":"priority-server",
        "ttl":86400000
    }
);
```

- This code will call 'myservice' sequence from 'myproject' and store data in local mobile cache for 86400000 mseconds.

SESSION MANAGEMENT AND ENHANCED COMMUNICATION CRYPTOGRAPHY

- SDK gives automatic SSL/TLS secured connections
 - Client certificate support
 - Authentication Cookies (SAML, Other) Support
- Optional Enhanced Cryptography by over-ciphering sent data to Convertigo MBaaS server. Cipher is done using AES256 encryption with Session specific Private key exchanged when session is established.
- Session Management is automatic. The SDK will maintain a Client session with MBaaS server with no need of managing session cookie manually.

- Sample code (Java Android)

```
C80 c8o = new C80("https://myserver/convertigo/projects/myproject",
    new C8oSettings().
        setTrustAllCertificates(true). //Trust self signed certs
        setTimeout(10000).           // Cnx timeout
        setUseEncryption(true).      // use over-ciphering
        addCookie("custom1", "value1"). // use custom cookie 1
        addCookie("custom3", "value3") // use custom cookie 2
);
```

- This code will establish a SSL session with a Convertigo MBaaS server with over-ciphering and custom cookies.

ASYNC CALLS AND UI TREAD MANAGEMENT

- In all Mobile OS calls to the network should be done asynchronously to prevent blocking the UI thread.
- This work is done automatically by the SDK
- A promise API handles async callbacks and can automatically switch to UI threads to enable programmers to update the UI with data returned from the MBaaS Server.

- **Sample code (Java Android)**

```
c8o.callJson("project.sequence").  
  thenUI(new C8oOnResponse<JSONObject>() {  
    @Override  
    public C8oPromise<JSONObject>  
      run(JSONObject response, Map<String, Object> parameters) throws Throwable {  
      // Update UI here  
  
      .....  
    }  
  });
```

- This code will call asynchronously the "sequence" in project "project" and when the data is returned, will execute the C8oOnResponse handler in the UI thread giving the opportunity to update the UI.

SERVER SIDE EVENT LOG MANAGEMENT

- Monitor on Convertigo Server all Mobile client activity
- Any event on the client side can be logged and centralized in the server.
- Log Levels can be set dynamically at run time by the server admin
- Client logs can be searched, filtered by device ids, or any other criteria using the Convertigo server admin console.
- Simple cross-platform API to log an event.

- Sample code (Java Android)

```
c8o.log(C8oLogger.DEBUG, "User has clicked subscribe button");
```

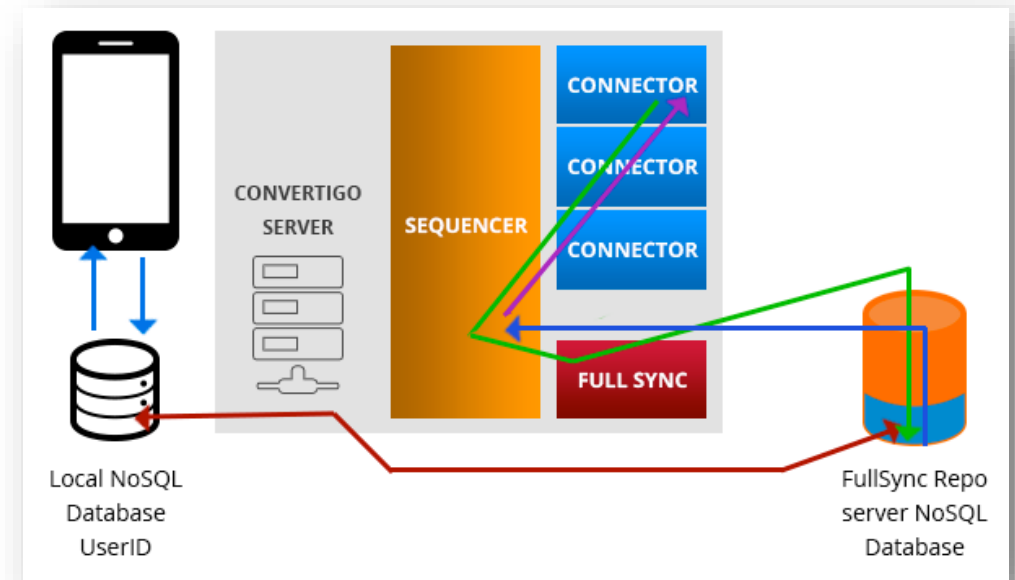
- This code will log the message to the server according to the log level. Logs levels can be configured dynamically on the Server console.

ERROR MANAGEMENT









- HTTP protocol errors (500, 404) and network errors are automatically handled by an error handlers. No need to handle these errors manually.
- Functional errors are received as standard JSON objects
 - Programmer can take decisions by looking JSON object keys.
- Network presence is automatically handled to manage local cache


OPTIONAL FULL SYNC MANAGEMENT


- SDK can also manage all the FullSync Synchronization on an local NoSQL database
- Calls the local database follows the `c8o.CallJSON` api with special "requestables"
 - `"fs://database.get"` to get an object from the local database named "database"
 - `"fs://database.put"` to put an object in the local database named "database"
 - `"fs://database.view"` to query a view from the local database named "database"
 - `"fs://database.sync"` to synchronize pull all the data for this user from Convertigo MBaaS to this database and to push all local data back to server.
 - ...



SDK IS AVAILABLE FOR ALL MOBILE CLIENT PLATFORMS

Platform	Languages (IDE)		
iOS	C# (Xamarin) 		Objective-C (Xcode)  Swift (Xcode) 
Android	C# (Xamarin) 		Java (Android Studio) 
Windows Phone	C# (Xamarin)	C# (Visual Studio)	
Windows Universal App	C# (Xamarin) 	C# (Visual Studio) 	
Windows WPF (Desktop)		C# (Visual Studio) 	

 FullSync Available in SDK 2.0

 FullSync Planned