IOS Softphone Development

MizuTech iOS Softphone VM, project and source code guide

Contents

About

MizuPhone for iOS is a simple to use SIP softphone for iPhone, iPod and iPad devices based on PJSIP and Siphon. The iOS SIP softphone can be used with in any SIP networks, with any SIP server, PBX or softswitch.

Aside from the public and customized versions, we can also provide the source code to our customers as a ready turnkey project so you can further customize, modify or improve the softphone after your needs.

In this guide, we will provide all the necessary information on how to start development with the iOS softphone.

License terms

The Mizutech iOS softphone source code is a confidential material.
We (Mizutech SRL) provide the source code of the iOS softphone for your organization (as an individual, company or any organization), under the following terms:

- Title, ownership rights, and intellectual property rights in the Software shall remain with MizuTech and/or its suppliers.
- The agreement and the license granted hereunder will terminate automatically if you fail to comply with the limitations described herein. Upon termination, you must destroy all copies of the Software
- You can use the source code to develop your own iOS application.

- We reserve all rights regarding the source code. The iOS source code constitutes valuable copyrighted intellectual property and you will not become the owner of the software.
- We can provide the source for your organization only as a confidential material to fulfill your own app development requirements.
- You are allowed to use the source code only for your organization internal needs. Any kind of distribution of the source code is strictly forbidden.
- No right is granted to sell, resell, rent, distribute, make available, publish or otherwise transfer the source code. The source code must never leak out of your organization.
- The softphone is based on PJSIP, which is licensed under GPL or proprietary. By using the source code or the virtual machine provided by Mizutech, you must also agree with the PJSIP license terms.
- You shall not employ source code in any way that competes either directly or indirectly with us including but not limited to creation of derivative works that compete either directly or indirectly with our MizuPhone software.
- You shall not sell or offer as a service the derivative works for other companies, organizations or individuals.
- The source code and the virtual machine is provided "as is" without any warranty of any kind

By opening or coping the source code files, you agree with all the above terms.
Otherwise, you must immediately delete all copies of the MizuTech virtual machine and all the source code from all your computers.

Requirements

1. Familiarity with MacOS, Objective-C and iOS development knowledge
2. MacOS virtual machine image from Mizutech with the turn-key MizuPhone project
3. Alternatively, your own MacOS with a new version of XCode (which is often required for publishing) might not work on older versions of Mac OS.
    a. Register an Apple Developer Account.
    b. Download and install the latest version of XCode. This will install also the latest version of iOS SDK.
    c. Download the iOS softphone project source
4. Open XCode and import the project: File menu -> Add Files to… -> locate and open “MizuPhone.xcodeproj” file.

Virtual Machine

This chapter is useful only if you received a developer virtual machine image. Otherwise, please skip this chapter.

MizuTech can provide a turnkey virtual machine with everything configured for the iOS softphone development.

This section describes how to configure and launch the development environment VMware image. The virtual machine contains a completely setup and working environment for iOS softphone development; just download the VMware image, run it and you are ready to develop and publish iOS softphone to App Store.

Prerequisites

- A PC running windows
- VMware Player or VMware Workstation
- Developer Mac image form Mizutech

If you do not like VMware, you might convert the virtual machine image to other format such as VirtualBox, but that might require some additional tweaking to work with the MacOS guest and in this guide, we are using VMware only.

Download the virtual machine image first from Mizutech. If compressed, unzip it (usually compressed with zip).

The steps required to run the MacOS developer virtual machine are described below:

Install VMware

Follow the steps below on your host PC:

1. Install the VMware player or workstation
2. Unzip the Developer Mac image downloaded form Mizutech
3. Patch VMware to be able to run Mac OS:
    a. Make sure VMware is closed

b. Navigate to “vmware-unlocker” which can be found in the downloaded image
c. Right click on “win-install.cmd” -> Run as administrator
d. Wait a few minutes until the operation is finished (the command line widow will close)

Create VM

Follow the steps below to create the guest MacOS virtual machine:

1. Launch VMware and click *File* menu -> *New virtual machine.*
2. On the first page of the wizard select *Typical* and click *Next.*
3. On the *Guest Operating System Installation* page click *I will install the operation system later* and click *Next.*
4. On the *Select a Guest Operating System* page select *Apple Mac OS X* for the operating system and select *macOS 10.14* form the dropdown list. Click *Next*.
5. On the *Name the Virtual Machine* page give a name to your machine and select the location for this machine. Click *Next*.
6. On the *Specify disk capacity* page just click *Next*. We will replace this disk anyway in the following steps.
7. Click *Finish*.

VM Configuration

Follow the steps below to edit the virtual machine settings:

1. Go to *VM* menu -> *Settings*
2. On *Hardware* tab:
    a. Select *Memory* and give it at least 4GB of system memory.
    b. Select *Processors* and set the number of processors and processor cores based on your hardware
    c. Select *USB Controller:* for *USB Compatibility* select *USB 2.0* and check *Show all USB input devices* checkbox
3. On *Hardware* tab select *Hard Disk*:
    a. Click *Remove*.
    b. Click *Add*.
    c. On *Hardware Type* page select *Hard Disk* and click *Next*.
    d. On *Select a Disk Type* page leave the recommended settings and click *Next*.
    e. On *Select a Disk* page select *Use an existing virtual disk* and click *Next*.
    f. On *Select an Existing Disk* page click *Browse* and select the virtual machine that you have downloaded from Mizutech and click *Finish*.
    g. If a popup appears asking “Convert existing virtual disk to newer format?”, just click “Keep existing Format”
4. Click *OK* on the bottom of the Settings page.
5. Navigate to your virtual machine files on your hard disk and find the file with extension “vmx”
    a. Open this file with Notepad
    b. Go to the end of the file and add the following text in a **new line**: smc.version = "0"
    c. Save the file and close it.
6. Adjust the Network and other settings after your needs.

We have finally finished setting up the Mac OS development virtual machine. Just start the virtual machine.
For the MAC machine the login (admin) username is “Mizutech” and the password is “mizutech”.
We do not recommend installing any upgrades as this might result in unbootable virtual machine - unless if new OS or XCode version is enforced by Apple. We also upgrade the virtual machine image usually once a year.

Development

After the virtual machine started and you have logged in using “mizutech” password we are ready to start development. A shortcut to the MizuDemo and or to MizuPhone iOS softphone project directory can be found on the desktop or under the Apps shortcut.

Note: files and folders name might be slightly different depending on the source code type (softphone source, SDK source, etc) and version.

Go to project directory and open MizuDemo.xcodeproj.

MizuDemo can be installed and launched from XCode on an iOS device or on iOS Simulator.

Launch on device

1. Connect your iOS device to you PC using your USB-Lightning cable
2. In VMware click *VM* in the top menu bar -> *Removable devices* -> *Apple iPhone* -> click *Connect(Disconnect from host)*
3. In XCode select your device and click Run.

Launch on Simulator

1. To run on Simulator we need to change a few things:
    a. Go to MizuDemo project directory and rename *pjproject-svn_X64Simulator* to pjproject-svn (rename original *pjproject-svn* to something else)
    b. In XCode click on MizuDemo project in the left pane, go to *Build Settings* and change *Valid Architectures* from ***armv7 arm64*** to **x86_64**
2. Select the Simulator in the top-left corner in XCode and click Run.

Do not forget to reverse the above described changes to be able to run the project on a device or to be able to publish it.

Project structure and dependencies

The project consists of the modified and prepared PJSIP communication library and the user interface.

Folders

The most important folders are the followings:

- Apps: root folder
- pjproject-svn: PJSIP library built for iOS (the ARM version by default)
- pjproject-svn_X64Simulator: PJSIP for x86/x64 (rename to pjproject-svn if you need this)
- Siphon2: contains all the GUI related files and resources (if not exists, then the Apps folder)
- Classes: source code, required
- Mizu: source and icons
- MizuPhone_SDK.xcodeprj: the test project to be opened
- Other: pjsip API
- Resources: project resources
- Sounds: sound resources
- Settings: settings app (will generate Settings.bundle; not needed)
- Settings.bundle: settings app
- SiphonSettings.xcodeproj: settings app project
- MizuPhoneNew: generated app icons and launch images
- build: auto generated (not relevant, might be missing)
- G729: not needed but can be kept if later modifications are required
- GoogleMobileAdsSDK: not needed but can be kept if later modifications are required
- Info.plist: test project plist

The pjproject-svn and the GUI project (Siphon2 or other) folders must be in the same directory, otherwise you will have to change the paths to PJISP library.

Note: You might not receive all these folders or the folder names might be slightly different in your deliverable, depending on your license, order and version.

SDK

For the SIP stack, the PJSIP library is used. The library is compiled for iOS arrmv7 and arm64 architectures. G729 and Opus codecs and other patches are also added to the library.

A detailed SDK specific guide can be found here.

GUI

As mentioned above, all the user interface related files are located inside the Apps folder, usually in the “Mizu” or “Siphon2” directory. This is bundled in an XCode project named: “*MizuPhone.xcodeproj*”. The project is written in Objective-C and C languages.

Compatibility

The softphone is backward compatible including iOS 8. It can run on both armv7 and arm64 architecture devices.

Build and run

Connect an iOS device to your Mac and click Build and Run button. The application will be installed on the device. On first start, the Setup page will be displayed where the user can enter his/hers SIP login details.

Publish

You can publish MizuDemo or MizuPhone under your own brand name (MizuDemo and MizuPhone is already used by MizuTech, rename your app to your own brand name).

Project settings

- target: armv7 amr64
- def_custom.h (config)
- SettingsForCustomized.bundle / SettingsMizuPhone.bundle -> rename to Settings.bundle
- XCode -> Preferences -> Accounts -> add customer (your) account
- Project -> Target -> Signing Capabilities (All; always All)
- Product -> Archive -> Publish

AppStore publishing

1. Create an Apple Developer account, if you don’t already have one, and enroll in the iOS Developer program.
2. Create Distribution certificate, App ID and Distribution Provisioning profile on the developer.
3. Go to XCode Menu bar -> XCode -> Preferences -> Accounts and add you developer Apple ID here.
4. Build the iOS softphone with XCode: Menu bar -> Product -> Archive.
5. Upload and publish the softphone with App Store Connect

Certificates

You can find the certificates at developer.apple.com -> certificates, identifiers & profiles:

- iOS Distribution: this is the most important to sign the app
- Developer ID Application and Installer -> required for MacOS
- iOS Development: for testing to be able to install app to my own device
- Apple push services: for the VoIP server or push gateway (not for XCode or for app)
- VoIP Services: for better push (required for voip apps)

A more in-depth guide about publishing your iOS softphone can be found here (see the “Publish” chapter).

GUI architecture

The project consists of the “SiphonApplication” app delegate class and several UIViewController classes. Each UIViewController class has its own standalone Interface Builder xib file. No storyboard is used in this project.

Navigation between view controllers is achieved using “UINavigationController”.

On first start the “Setup” view controller is launched. On subsequent launches, if there SIP login details were entered, the Numpad VC will be displayed.

Customizations

The most important parameters can be set in “def_custom.h” header file. Below is a list of all parameters:

```
#define [BRAND NAME]

#define CUSTOM_COMPANY_BRANDNAME @"BRAND_NAME" // softphone brand name
#define CUSTOM_COMPANY_VERSION_NAME @"3.4.0329" // version number
#define CUSTOM_COMPANYNAME @"COMPANY_NAME" // your company name
#define CUSTOM_COMPANY_SERVERADDRESS @"yourdomain.com"

// the theme colors definitions
#define TEXT_COLOR @"#FFFFFF"
#define BACKGROUND_COLOR @"#001a22"
#define HEADER_BG @"#001a22"
#define PHONE_FIELD_BG @"#22393f"
#define PHONE_FIELD_HINT_COLOR @"#989898"
#define NUMPAD_BG @"#22393f"
#define NUMPAD_BTN_BG @"#001a22"
#define NUMPAD_BTN_BG_PRESSED @"#22393f"
#define NUMPAD_BTN_TEXT_COLOR @"#39caff"
#define SEPARATOR_BG @"#001a22"
#define CALL_BTN_BG @"#22393f"
#define CALL_BTN_BG_PRESSED @"001a22"

// page displayed once or on every start; can by a URL or text; usually used for privacy policy
#define STARTPAGE_URL @"" // https://domain.com/my-start-page.html
#define STARTPAGE_TITLE @"" // title
#define STARTPAGE_TEXT @"" // text to be displayed; used instead of URL
#define STARTPAGE_TYPE 0 // 0: display once with OK button, 1: display once with confirmation, 2: display always

#define QRCODE_LOGIN 2 // QR code login feature: 0=disabled, 1=enabled, display QRcode button only if no username/password is set, 2=enabled, always display QRcode button on Settings page

#define LOGO_ACTIONBAR
#define CALLKIT 1 // use CallKit integration
#define ENABLE_PUSHKIT 1 // 0=No - Disable, 1=Yes - Autodetect, 2=Yes - Direct, 3=Yes - via Gateway

//#define AUTO_CREATE_NEW_USER // if defined, it will automatically create a new user on first start. Requires CUSTOM_COMPANY_NEWUSER to be set
#define ABOUT_WEBVIEW_URL @"" // load URL in About view controller web view
//#define LOGO_ACTIONBAR   // if defined, displays logo on top of dial pad page

#define CUSTOM_COMPANY_DELAY_YEAR 2021 // show advanced functions (for example credit recharge) only after 1 month so it will NOT be rejected by Apple review
#define CUSTOM_COMPANY_DELAY_MONTH 6
#define CUSTOM_COMPANY_DELAY_DAY 1

#define HASCHAT // disables chat feature if not defined
#define CUSTOM_COMPANY_WEBLINK @""
#define CUSTOM_COMPANY_MAIL @""

#define CUSTOM_COMPANY_SERVERSALT @"" // define server salt if any; used for http request
#define CUSTOM_COMPANY_CREDIT_REQUEST @""
#define CUSTOM_COMPANY_RATING_REQUEST @""
#define CUSTOM_COMPANY_NEWUSER @""
#define CUSTOM_COMPANY_RECHARGE @""
#define CUSTOM_COMPANY_CALLBACK @""
#define CUSTOM_COMPANY_P2P @""
#define CUSTOM_COMPANY_SMS @""
```

GUI classes

The most important GUI classes are the followings:
“AboutViewController”: display information about the softphone and company
“CallBackViewController”: user interface for initiating callback
“CallViewController”: user interface of the in-call page
“HistoryViewController”: call history list page
“ChatViewController”: messaging page
“ChatInboxViewController”: messaging inbox list page
“ContactViewController”: contacts list page
“NewUserViewController”: new user registration page
“P2PViewController”: user interface used for initiating phone to phone calls
“P2PViewController”: main dial pad page
“RechargeViewController”: user interface for credit recharge
“SetupViewController”: login page
“SiphoneApplication”: the application delegate class of the softphone
“SMSViewController”: view controller for sending SMS
“QRcodeViewController”: used for reading QR codes and bar codes

Permissions

All permissions are specified in info.plist. Permissions are managed in different strategic locations in the source code, requested only when needed.

Settings

All setting parameters are stored using NSUserDefaults. User configurable application settings in native settings app are in “Settings.bundle” structure.

Code for saving a parameter to settings:

```
NSUserDefaults *userDef = [NSUserDefaults standardUserDefaults];
[userDef setObject:pushkit_token forKey:@"pushkit_token"];
[userDef synchronize];
```

Code for getting a parameter value:

```
NSUserDefaults *userDef = [NSUserDefaults standardUserDefaults];
NSString *username = [userDef stringForKey: @"username"];
```

Contacts list

The contacts list presented on Contact view controller is the native contact list.

A custom contact list is implemented which is capable to request a contact list from the server. It requires a define in the def_custom.h header file.
Example:
#define CUSTOM_CONTACTLIST @"http://www.myconytactapi.com/getcontacts"
All the common keywords can be used like for the usual API requests.
Contact request answer format should be JSON like in the below example:

```
{
  "result_code": 0,
  "time": "2019-07-03 11:20:57",
  "data": [{
                                        "user_name": "Matthew Rossi",
                                        "user_number": "03853067543"},
                                        {
                                        "user_name": "Paul Bianchi",
                                        "user_number": "03852145634"},
                                        {
                                        "user_name": "Erik Verdi",
                                        "user_number": "110"},
  ],
```

```
  "error": ""
}
#define SERVERADDRESSBOOK_URL @""
```

From online builder this can be configured with the serveraddressbook_url parameter.

SDK API

The softphone uses the PJSIP library.
A detailed SDK documentation can be found here.

Two-way communication with the SIP stack is achieved with the SIP stack API and a notification mechanism.

The API

PJSIP API functions are called from “call.m” class.

Notifications

Notifications about the SIP stack states are received using NSNotificationCenter.

For example for receiving register related statuses we implement the “on_reg_state()” callback function from the PJSIP API and send notifications to main thread using NSNotificationCenter:

```
[[NSNotificationCenter defaultCenter] postNotificationOnMainThreadWithName:kSIPRegState
                                                                     object:nil
                                                                   userInfo:userinfo];
```

Important tasks

Push notifications

PushKit is used for VoIP push notifications.

On iOS, applications are not allowed to run in background, so to be able to receive incoming calls and messages, MizuPhone uses PushKit notifications. FCM notifications can also be added to MizuPhone softphone. For the complete guide for push notifications and background modes, please check out this guide.

To enable pushkit, you will need to set the ENABLE_PUSHKIT variable: 0=no,1=yes,2=direct,3=gateway

CallKit

CallKit is used for native dialer integration.
You might enable/disable CallKit with the CALLKIT define.
For more details, please visit this link:
https://developer.apple.com/documentation/callkit

In-App Purchase

Configure iOS In-App Purchase and purchasable products (credit recharge) for your application on iTunes Connect.
Send us the exact product IDs.

Enable in softphone at build time:

- In def_custom.h add #define INAPP_PURCHASE
- In def_custom.h specify InApp Purchase product ids separated by comma in INAPP_PURCHASE_PRODUCT_IDS.

Example:
#define INAPP_PURCHASE_PRODUCT_IDS @"com.domain.sip.recharge.5.00,com.domain.sip.recharge.10.00,com.domain.sip.recharge.20.00"

Error handling

This section is about exceptions and error handling

Some of the Objective C code ins placed inside a try-catch blocks, but in Objective C error handling is not so straight forward. Only some types of exceptions are caught by the try-catch blocks.

To log any messages, use the following static functions located in *SiphoneApplication:*

```
+ (void)PutToDebugLog: (int)level file:(NSString *)name msg:(NSString *)message
```

```
[SiphonApplication PutToDebugLog:3 file: @"MyFile" msg:@"log any message"];
```

The level parameter specifies the importance of the logged messages ranging from 1 to 6.

All these messages are saved into a log file and in the application internal storage. The log file can be sent to support by going to “Setup page -> Send Log”.

Logging can enabled/disabled from Settings app -> MizuPhone -> Phone settings -> Debugging log file.

Logs are sent in email and by default, all log files are sent to Mizutech. You can change the email address to which logs are sent with the following NSString define inf def_custom.h: `#define LOGMAIL_ADDRESS @"email@domain.com"`.

Customization/branding

All the customization details can be set with defines in *def_custom.h* header file located in \Mizu\ directory. You will find there a customization template with all the defines and comments next to them for explanation.

The application icons are located in \MizuPhoneNew\Images.xcassets\AppIcon.appiconset\ and the following size icons are needed:

- Icon.png - size is 57 pixels
- Icon20x2.png
- Icon20x3.png
- Icon29x2.png
- Icon29x3.png
- Icon40x2.png
- Icon40x3.png
- Icon60x3.png
- Icon120.png
- Icon1024.png

The application launch images are located in \MizuPhoneNew\Images.xcassets\LaunchImage.launchimage\ and the following sizes are needed:

- Default.png -size: 320 x 480 pixels
- Default@2x.png -size: 640 x 960 pixels
- Default@3x.png -size: 1242 x 2208 pixels
- Default-568h@2x-1.png -size: 640 x 1136 pixels
- Default-568h@2x-2.png -size: 640 x 1136 pixels
- Default-667h@2x.png -size: 750 x 1334 pixels

Important note: all icons and launch images must be in .PNG format and must NOT have transparency!

The color theme of the softphone is also set in *def_custom.h* with the following defines, where you can change the colors:

```
#define TEXT_COLOR @"#FFFFFF"
#define BACKGROUND_COLOR @"#001a22"
#define HEADER_BG @"#001a22"
#define PHONE_FIELD_BG @"#22393f"
```

```
#define PHONE_FIELD_HINT_COLOR @"#989898"
#define NUMPAD_BG @"#22393f"
#define NUMPAD_BTN_BG @"#001a22"
#define NUMPAD_BTN_BG_PRESSED @"#22393f"
#define NUMPAD_BTN_TEXT_COLOR @"#39caff"
#define SEPARATOR_BG @"#001a22"
#define CALL_BTN_BG @"#22393f"
#define CALL_BTN_BG_PRESSED @"#001A22"
```

Change the login page (setup page)

The setup page consists of the *SetupViewController* class and it's Interface Builder xib file *SetupViewController.xib* where you can change the structure of the page as you wish.

Change the main page (num pad)

The Numpad page consists of the *PhoneViewController* class and it's Interface Builder xib file *PhoneViewController.xib* where you can change the structure of the page as you wish.

Add a new setting

All setting parameters are stored using NSUserDefaults.

Code for saving a new or existing parameter:

```
NSUserDefaults *userDef = [NSUserDefaults standardUserDefaults];
[userDef setObject:pushkit_token forKey:@"pushkit_token"];
[userDef synchronize];
```

Code for getting a parameter value:

```
NSUserDefaults *userDef = [NSUserDefaults standardUserDefaults];
NSString *username = [userDef stringForKey: @"username"];
```

The application settings located in native Settings applications are located in "Settings.bundle" structure. Here you can add/remove any settings you wish.

Remove a setting

Use NSUserDefaults and rewrite the settings value to nil or whatever value you wish.

Change launch image

The application launch images are located in \MizuPhoneNew\Images.xcassets\LaunchImage.launchimage\ and the following sizes are needed:

- Default.png -size: 320 x 480 pixels
- Default@2x.png -size: 640 x 960 pixels
- Default@3x.png -size: 1242 x 2208 pixels
- Default-568h@2x-1.png -size: 640 x 1136 pixels
- Default-568h@2x-2.png -size: 640 x 1136 pixels
- Default-667h@2x.png -size: 750 x 1334 pixels

Important note: all icons and launch images must be in .PNG format and must NOT have transparency!

Translate

All text resources are stored in "Localizable.strings" files. These are located in the Resources folder (or at \MizuPhoneNew\Resources).

There are separate "Localizable.strings" files for every language/locale.

Test

The softphone can be tested by running it directly from XCode on any iOS device.

The final distribution version can be tested using TestFlight:

- upload the softphone to iTunesconnect
- install TestFlight app form App Store on your iOS devices
- invite testers (from iTunesconnect) to test the softphone

Prepare VM

This section is for mizutech support only about how to prepare the MacOS development virtual machine. You can skip this point.
Project location: at Documents\Apps\MizuPhoneNew\Siphon\ (or at Documents\Apps\)

- Siphon2 (or the Apps root folder)
- pjproject-svn (the SDK)
- delete all other files and folders

Prepare source code for the distribution (the order of "replace" is important):

1. Remove all comment lines starting with: //--
2. Remove all code between comments: //OPSSTART and //OPSEND
3. Remove all code between comments (branding specific): //BRANDSTART and //BRANDEND
4. Remove all lines containing (case sensitive) AKOSDEBUG
5. Remove these comments (just uncomment this line) //USEDFOROPS
6. Delete files: SettingsForCustomized.bundle and SettingsMizuPhone.bundle

MBuilder

This section contains only technical details useful for debugging the automated build process by Mizutech support. Please ignore this section.

Configuration file:

- configuration file should be named: *iosmbuild.txt*
- it will be downloaded from internal mbuild web ftp service.
- the content should be the full configuration file (the same as displayed in MTasks -> MBuilder window)
- in the *#----------BOF EXTRA SETTINGS----------* section of the config file the following parameters must be set:

```
Common.packagename = "com.domain.brand.app"; // must be set
Common.iosbrandname="MyPhone"; // must be set
Common.iospackagename="com.domain.brand"; // must be set
Common.developerteamid="xxx"; // must be set; login to https://developer.apple.com and go to Membership -> Team ID
Common.iosprovisioningprofilename="MbuilderDistribution.mobileprovision.zip"; // must be set
Common.iosprovisioningprofiletype="app-store"; // must be set
Common.iosdistributioncertificatename="mizu_distribution.cer.zip"; // must be set
```

Parameters are loaded only from
#----------BOF JAVA CODE---------- es #----------EOF JAVA CODE----------
#----------BOF EXTRA SETTINGS---------- es #----------EOF EXTRA SETTINGS---------- kozott van.

Set the Common.mbuildoption_sourcecode option after your needs:
1=whole source with vmware, 2=SDK only with vmware, 3=GUI only with vmware, 4=whole source, 5=SDK only, 6=GUI only

iOS MBuilder result:

After build has finished, one of the following text messages will be written to *iosmbuild.txt*

- OK: build succeeded: [BRANDNAME] [VERSION]

- ERROR: build failed: [BRANDNAME] [VERSION]

Required resources:

All file type resource must be available for download at the internal mbuild web ftp service.

1. mbuilder config file: *iosmbuild.txt*
2. [BRANDNAME]_icon.png -size: 1024x1024 pixels
3. [BRANDNAME]_logo.png -size: minimum 250 pixel width
4. [BRANDNAME_launch.png] -splash screen launch image with exact size: 1242x2208 pixels
5. "isopackagename" config parameter must be specified. This package name must be the same as the "Bundle Identifier" specified in the "App ID" configured on Developer portal.
6. "iosdeveloperteamid" should be passed in config file. Log into the Developer Portal using the Apple ID for your Apple Developer Account and look under the "Membership" section “Team ID”.
7. "iosprovisioningprofilename" the name of the zipped distribution provisioning profile
8. " iosprovisioningprofiletype" possible values: "app-store", "ad-hoc"
9. "iosdistributioncertificatename" the name of the zipped iOS distribution certificate
10. If PushKit notifications will be used then:
    - Login to the Apple Developer portal.
    - Go to “Certificates, Identifiers & Profiles” -> “Certificates” and generate a “VoIP Services“ certificate.
    - Go to “Certificates, Identifiers & Profiles” -> “Identifiers” -> click on your “App ID” -> “Capabilities” and enable “Push Notifications”. Make sure that there are no conflicts with Push Notifications certificates here.

Required resources for Ad-Hoc (for demo version):

1. mbuilder config file
2. "iosprovisioningprofilename" the name of the zipped **Ad-Hoc** provisioning profile
3. "iosprovisioningprofiletype" set to: "ad-hoc"
4. "iosdistributioncertificatename" the name of the zipped provisioning profile
5. UDIDs of the test devices: You can find your UDID by connecting your iOS device to your computer -> start iTunes -> select your device -> then Summary tab on the left.
   In the first "box" you will see a field called "ECID:" or "Serial Number", click it a few times, until you see your UDID, then right click an copy.

Compile the MBuilder executable

- XCode -> open the Mbuilder project and make any required changes then build.
- Product -> archive -> export to: apps -> mbuilder -> output -> mbuilder

The SDK build process

Open the MBuilder.xcodeproj file in Xcode

1. All the important logic is in the AppViewController.m file
2. Downloads the iosmbuild.txt-t (and all other resources such as logo, icon) and pass the config file content to the "BuildConfigFile" function
3. If "mbuildoption_sourcecode" parameter is found then call the "InitMbuildSource" function
4. Check what type if source code is requested and run the "copy_mbs.sh" shell script which will copy the required source code to the "word" directory
5. Call the "JustSource" function which will replace everything. Then call the "archive_mbs.sh" which will zip it.
6. The the "FTPUploadFile" function will be called which will upload it.

Test the MAC MBuilder

You can use the MTSoftphone app to test the mbuilder process

1. copy the required resources to the /WebFTP_G/... ftp location
2. launch the MBuilder app

3. wait to finish the app build
4. use the "Transporter" app to upload it to the App Store Connect

Other options
SCURL_ONSTART
SCURL_ONINCALLSETUP
SCURL_ONOUTCALLSETUP

iOS push notification:
ISMIZUSERVER
ENABLE_PUSHKIT 0=no,1=yes,2=direct,3=gateway
ENABLE_PUSHKIT set from autoprov from enablepushkit

ability to easily disable features (chat, etc) with the same disableoptions parameter format
These have to be handled by mbuilder using defines:
#define HASCHAT
#define HASCONFERENCE

autogenpassword (implemented in Android. add also to iOS and webphone)
#define AUTOGENPASSWORD 0
0=disabled
1=enabled, password is user editable
2=enabled, password is not editable

password recovery page (configurable URL) -what is the url? (String forgotpasswordurl = "";)
#define FORGOTPASSWORDURL @""

Accepted parameters:

- useencryption
- fastencryptkeystr
- tunnelserveraddress
- cust_serverdomain
- preconfig_serveraddress
- haschat
- autoprovisioning
- webphoneautoprov
- autocreatenewuser
- serversalt
- cust_creditrequest
- cust_ratingrequest
- cust_newuser
- cust_newuser_second
- cust_recharge
- cust_callback
- cust_p2p
- cust_sms
- startpage_title
- startpage_url
- startpage_text
- startpage_type
- serveraddressbook_url
- enablefcm
- enablecallkit
- autogenpassword
- messagepopup
- logmanualemail

- autocreatenewuser
- showlogoinactionbar
- keepaliveival
- callbacknumber
- proxyaddress
- transport
- mediaencryption
- dtmfmode
- voicemailnum
- use_pcmu
- use_pcma
- use_opuswb
- use_speexwb
- use_ilbc
- use_gsm
- use_g729
- prefcodec
- signalingport
- rtpport
- loglevel
- stunserveraddress
- showserverinput
- qrcode_login

- bgcolorheader
- bgcolor
- scrollbarcolor
- listbgcolor
- listitembgcolor
- buttoncolor
- buttonhover
- buttonbordercolor
- tabcolor
- tabselectedcolor
- bgdialpadnrfield
- bgdialpadbtn
- bgdialpadfooter
- fontctheme
- fontcwhite
- fontcstatus
- fontfamily

Miscellaneous

Random/quick notes here:

Compile or build errors:
In case of any error(s) appear at build time, they will be displayed in XCode in the left pane in *Issue navigator*.

use lipo.sh to combine arm7 and arm64

to run in simulator:
change MizuPhone -> Targets -> MizuPhone -> Build settings -> Valid architetures x86_64

publish:
target: armv7 amr64
def_custom.h (config)
SettingsForCustomized.bundle / SettingsMizuPhone.bundle -> rename to Settings.bundle
XCode -> Preferences -> Accounts -> add customer acc
Project -> Targer -> Signing Capabilities (All; always All)
Product -> Archive -> Publish

developer.apple.com -> certicates, identifiers & profiles

iOS Distribution: this is the most important to sign the app
Developer ID Application and Installer -> required for MacOS
iOS Development: for testing to be able to install app to my own device
Apple push services: for push server (not for xcode or for app)
VoIP Services: for better push (required for voip apps)

VM Cleanup

The used disk space by your MacOS guest virtual machine might grow with time.
You can compact/shrink in the following way:

- On the VMWare guest (inside the Mac OS virtual machine):
  - sudo pmset hibernatemode 0; sudo rm -f /var/vm/sleepimage
  - cat /dev/zero > wipefile; rm wipefile
    - this might take a few minutes to finish
  - you might defragment the "Macintosh HD" partition inside your VM
  - shut down
- From the host:
  - vmware-vdiskmanager -k <Fully Qualified Pathname to the Virtual Hard Disk .vmdk file>
    - on Windows the vmware-vdiskmanager.exe can be found at: C:\Program Files (x86)\VMware\
  - or if you are on linux:
    - sudo /Library/Application\ Support/VMware\ Tools/vmware-tools-cli disk shrink /
    - vmware-tools-cli disk list to see the available locations for wipe
    - vmware-tools-cli disk wipe <location> to wipe each location (repeat multiple times), without the shrink operation
    - vmware-tools-cli disk shrinkonly to do the final shrink operation.

See this page for more details:
https://communities.vmware.com/message/2399910

Objective-C

You will need Objective-C knowledge for the softphone development (and pure C if you wish to make changes in the PJSIP SDK).
Here are some Objective-C resources:
links:

books:
https://www.tutorialspoint.com/objective_c
https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html
https://www.ios-blog.com/tutorials/objective-c/beginners-ios-development-objective-c/
https://www.binpress.com/learn-objective-c-24-days/

gui:
https://www.raywenderlich.com/2829-objectively-speaking-a-crash-course-in-objective-c-for-ios-6
https://developer.apple.com/library/archive/documentation/General/Conceptual/CocoaEncyclopedia/Introduction/Introduction.html
https://en.wikipedia.org/wiki/Cocoa_(API)

chat sheet:
https://koenig-media.raywenderlich.com/downloads/RW-Objective-C-Cheatsheet-v-1-5.pdf
https://github.com/iwasrobbed/Objective-C-CheatSheet
http://cocoadevcentral.com/d/learn_objectivec/
https://www.ios-blog.com/tutorials/objective-c/objective-c-cheat-sheet/

from C/C++:
https://www.agiledeveloper.com/presentations/objective_c_for_experienced_programmers.pdf
https://www.codeproject.com/Articles/770577/From-Cplusplus-to-Objective-C-A-quick-guide-for-pr
https://pierre.chachatelier.fr/programmation/fichiers/cpp-objc-en.pdf
https://www.raywenderlich.com/2484-introduction-to-c-for-ios-developers-part-1
https://www.sitepoint.com/using-c-and-c-in-an-ios-app-with-objective-c/

Foundation Framework:
(standard library)

strings: NSString, NSScanner, NSCharacterSet
arrays: NSArray, NSDictionary, NSSet
date: NSDate, NSTimeZone, NSCalendar
errors: NSException, NSError (domain/code/userinfo)
files: NSFileManager
url's, sockets

sample:

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

@interface SampleClass:NSObject  //create an interface. It inherits NSObject, which is the base class of all objects.
   int length;    //private member variable
   -(double) volume; //public member variable
   @property(nonatomic, readwrite) int height;  //property will male height public (nonatomic/atomic, readwrite/readonly,
strong/unsafe_unretained/weak)
   - (void)sampleMethod;  //declare a method
@end  //end of an interface

@interface SampleClass() {  //extension to add private methods and private variables
  NSString *internalID;
}
```

class:

```
@implementation SampleClass  //implement the interface SampleClass
   @synthesize height;
   -(id)init {  //ctor
     self = [super init];
     length = 1;
     return self;
   }
   -(double) volume {  //???
     return length*breadth*height;
   }

   - (void)sampleMethod {   //implementation the sampleMethod method
     NSLog(@"Hello, World! \n");  //function available in Objective-C causes the message to be displayed on the screen
   }
@end

int main() {  //program execution begin here
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
  SampleClass *sampleClass = [[SampleClass alloc]init];
  SampleClass *sampleClass2 = [[SampleClass alloc]init];
  [sampleClass sampleMethod];
  int volume = [sampleClass volume];
  NSLog(@"Volume of Box1 : %f", volume);
  [pool drain];
  return 0;
}
```

protocol:

```
@protocol ProtocolName
@required
// list of required methods
@optional
// list of optional methods
@end

@interface MyClass : NSObject <MyProtocol>
...
@end
```

functions:

```
- (return_type) method_name:( argumentType1 )argumentName1
joiningArgument2:( argumentType2 )argumentName2 ...
```

```
        joiningArgumentn:( argumentTypen )argumentNamen {
          body of the function
        }

        - (int) Add:(int) num1 secondNumber:(int) num2 {
            //"secondNumber" is just a label
            return num1+ num2;
        }

memory managegement:
        manual (MRR):
            "alloc", "new", "copy", or "mutableCopy"
            take ownership: retain
            relinquish / release
        automatic (ARC)
```

Support

We provide the source code “as-is” with no additional support included.
We do not include iOS development services, support services or consultancy services.

We always test the delivered source code and it compiles/builds correctly on our test machines generating a correctly working iOS SIP SDK.
Please send a detailed problem description to support@mizu-voip.com if you find any issue. We cannot guarantee direct support, but we always consider all bug reports and we release new versions periodically with improvements and bug fixes.

Resources

- Mizutech home page
- iOS MizuPhone home page
- Contact