



Magnet™ Android Developer Guide

2.0

Revision A

© 2013 Magnet Systems, Inc. All rights reserved. This manual, in whole or in part, may not be reproduced, translated, or reduced to any machine-readable form without the prior written approval of Magnet Systems, Inc. Magnet Systems, Inc. reserves the right to make any modification to this manual or the information contained herein at any time without notice.

MAGNET SYSTEMS PROVIDES NO WARRANTY WITH REGARD TO THIS MANUAL, THE SOFTWARE, OR OTHER INFORMATION CONTAINED HEREIN AND HEREBY EXPRESSLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE WITH REGARD TO THIS MANUAL, THE SOFTWARE, OR SUCH OTHER INFORMATION. IN NO EVENT SHALL MAGNET SYSTEMS, INC. BE LIABLE FOR ANY INCIDENTAL, CONSEQUENTIAL, OR SPECIAL DAMAGES, WHETHER BASED ON TORT, CONTRACT, OR OTHERWISE, ARISING OUT OF OR IN CONNECTION WITH THIS MANUAL, THE SOFTWARE, OR OTHER INFORMATION CONTAINED HEREIN OR THE USE THEREOF.

All other trademarks are trademarks of their respective owners. Any Magnet Systems patents granted or pending in the United States and other countries are protected by law.

Magnet Systems, Inc.

435 Tasso Street
Suite 100
Palo Alto, CA
94301

650-329-5904

info@magnet.com

1. About This Guide	1
Audience	1
Document Organization	1
Related Documents	2
2. Introducing Magnet Mobile App Server	3
Magnet Mobile Enterprise Server	4
Custom Controllers	4
Caching and Off-line Message Delivery	4
Persistence	4
Transactions	5
Service Integration	5
Third-Party Service Controllers	5
Magnet Mobile App Manager	6
Mobile App Management Console	6
Mobile App Store	6
Deployment	6
Developer Factory	7
3. Getting Started	9
Setting Up Your Repository Environment	9
Creating a settings.xml File	9
Declaring in the POM File	10
Setting up Your IDE	11
Setting up Your Account in the Magnet Developer Factory	11
4. Managing Your Project with Developer Factory	13
Creating a Project	14
Displaying the Generated Assets	21
Modifying a Project	22
Uploading a Project	24

Removing a Project	25
Inviting Other Developers	27
5. Building your Android Project	29
Understanding the Project Structure	29
Building the Project with Eclipse	31
6. Defining Application Programming Interfaces	33
MagnetMobileClient	34
connect()	35
connect(String username, String password, String authority)	36
connect(com.magnet.mobile.android.ConnectionPreferences preferences)	36
disconnect()	37
getConnectionPreferences()	37
getConnectionService()	38
getConnectionState()	38
getService()	39
injectNow()	39
obtainPreferences()	40
registerConnectionListener()	40
startKernel()	41
stopKernel()	41
unregisterConnectionListener()	42
Enum MagnetMobileClient.Connection State	42
values	43
valueOf	44
ConnectionService	44
getBaseUri()	45
isAuthenticated()	45
isConnected()	45
isConnecting()	45
ConnectionServiceListener	46
onConnectionEvent()	46
Enum Constant	46
AUTHENTICATED	47
BAD_CREDENTIALS	47
BAD_VERSION	47



CONNECTED	47
CONNECTING	47
DISCONNECTED	48
UNAUTHENTICATED	48
EmbeddedMobileManager	49
getInstance()	50
getMessenger()	51
isFeatureAvailable()	51
isReady()	52
removeInstance()	52
MessageHandler	53
onClientRegistered()	55
onClientUnRegistered()	56
onError()	56
onMessageReceived()	57
onServerActivated()	57
onServerCancelled()	57
Messenger	58
activateClientDelivery()	60
activateServerNotification()	60
cancelClientDelivery()	61
cancelServerNotification()	61
getClientDeliveryToken()	62
getMessageHandlers()	62
getServerNotificationActivated()	62
isDeliveryTypeSupported()	63
isNotificationActivated()	63
onMessageReceived()	63
registerMessageHandler()	64
removeMessageHandler()	64
Enum Messenger.DeliveryType	65
toName	65
values	65
valueOf	66



MagnetActivity	67
initializeNow()	68
onCreate()	68
onSaveInstanceState()	68
setContentView()	68
setContentView()	68
setContentView()	68
shouldDeregisterSelfOnDestroy()	69
shouldInitializeOnCreate()	69
MagnetAndroidService	69
onCreate()	69
onDestroy()	70
shouldDeregisterSelfOnDestroy()	70
MagnetApplication	70
onCreate()	70
onTerminate()	70
AndroidSettingsConstants	71
MAGNET_DEV_FACTORY_CONFIGURATION_FILE_NAME	71
MAGNET_SHARED_PREFERENCE_FILE_NAME	71
PREFS_FILE_NAME	72
PREFS_KEY_DEFAULT_ENABLE_DATA_ENCRYPTION	72
PREFS_KEY_DEFAULT_ENABLE_DEBUG	72
PREFS_KEY_DEFAULT_ENABLE_LOCATION	72
PREFS_KEY_ENABLE_DATA_ENCRYPTION	72
PREFS_KEY_ENABLE_DEBUG	73
PREFS_KEY_ENABLE_LOCATION	73
PREFS_KEY_GCM_CLIENT_TOKEN	73
PREFS_KEY_GCM_SENDERID	73
PREFS_KEY_LOCATION_LAST_UPDATE_TIME	73
PREFS_KEY_PUSH_ACCOUNT_MAGNETURI	74
PREFS_KEY_USER_ME_MAGNETURI	74
7. Defining Caching and Off-line Mode Services	75
Controller	75
Requests	76
Reliable Requests	76
Async Requests	76



Listener	76
CallManager	77
cancelAllPendingCalls()	78
clearCache()	78
getAllPendingCalls()	78
getCalls()	79
reset()	79
run()	80
Options	80
Class AsyncCallOptions	80
create()	81
setCacheAge()	81
setConstraint()	82
setStateListener()	83
setToken()	83
Class ReliableCallOptions	84
create()	85
setCacheAge()	85
setCallTimeout	86
setConstraint()	86
setQueueName	87
setStateListener()	87
setToken()	87
Constraints	88
IsAllowed	88
AsyncController	89
Interface Call<T>	89
getId	90
getResultTime	90
getState	90
getToken	90
isResultFromCache	91
Enum Call.State	91
valueOf	92
values	93



Making a Reliable Request	93
Caching Results and Callback	94
Caching Results but no Callback	94
No Caching and No Callback	94
Making an Asynchronous Request	95
No Caching, Reliability, or Callback	95
No Caching or Reliability but With Callback	95
Caching and Callback, but no Reliability	96
Applying a Constraint	97
Code Samples	98
Injecting CallManager to Manage Calls	99
Using BroadcastReceiver to Change the State	101
8. Implementing Version Check Handler.....	103
9. Using Third-Party Social Controllers	105
Using the Salesforce Controller	105



The Magnet™ Android Developer Guide provides information and code samples for building enterprise native Android apps. It also provides instructions for uploading the generated project to a private cloud-based sandbox in the Magnet Developer Factory for testing.

Audience

This guide is intended for developers who write Android client-side apps that interact with the Magnet Enterprise Server applications. This document assumes that you are experienced in an android development environment and that you understand basic control flow of an application written in Java.

Document Organization

This guide includes these chapters:

- *Introducing Magnet Mobile App Server*, provides a high-level description of each component, its relationship, and interaction between the components.
- *Getting Started*, provides information that you need to gather before writing your app.
- *Managing Your Project with Developer Factory*, provides instructions for creating, modifying, and deleting your project by way of Magnet Developer Factory.
- *Building your Android Project*, provides information and instructions for building your project with Eclipse.
- *Defining Application Programming Interfaces*, provides API methods used for your project.
- *Defining Caching and Off-line Mode Services*, provides methods used for making reliable and asynchronous requests, and code samples for each request scenario.
- *Implementing Version Check Handler*, provides code for handling the VersionCheckException when the version in the Mobile Server and the Mobile App Manager is incompatible.

- *Using Third-Party Social Controllers*, provides sample code for using Salesforce and Facebook controllers.

Related Documents

The following table lists and describes other documents that are related to the Magnet products.

Document Title	Description
Magnet™ Server Application Developer Guide	Intended for developers who write Java-based server-side application, this guide provides information and code samples for building Magnet Enterprise Server applications.
Magnet™ iOS Developer Guide	Intended for iOS app developers, this guide provides information and code samples for building enterprise native iOS apps. It also provides instructions to upload the completed iOS app to your private sandbox for testing.
Magnet™ Mobile App Server Deployment Guide	Intended for IT administrators, this guide provides information and instructions for deploying the Magnet Mobile App Server to the Amazon cloud.
Magnet™ Mobile App Administrator Guide	Intended for IT administrators, this guide provides information and instructions for managing apps using the Magnet Mobile App Management Console.
Magnet™ Mobile for Android User Guide	Intended for users of Android devices, this guide provides information and instructions for installing the Magnet Mobile Server and using Apps@Work.
Magnet™ Mobile for iOS User Guide	Intended for users of iOS devices, this guide provides information and instructions for installing Setup@Work and using Apps@Work.

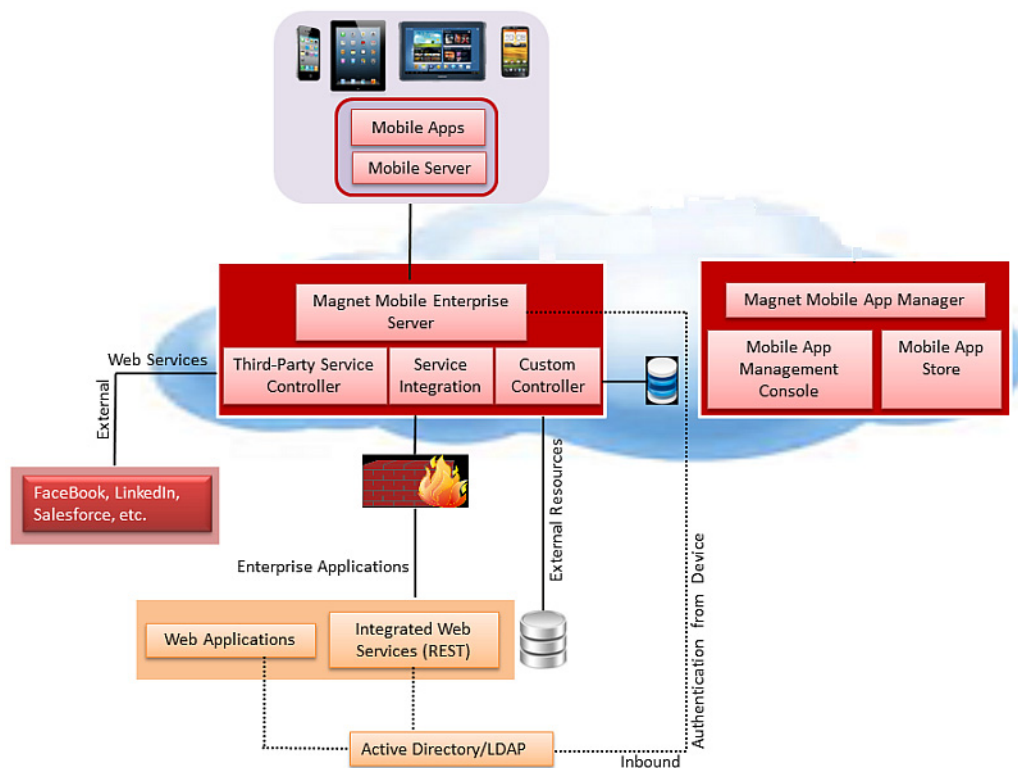


2. Introducing Magnet Mobile App Server

The Magnet™ Mobile App Server provides basic constructs for creating and manipulating app objects spanning the server and mobile app contexts. The Magnet Mobile App Server:

- Enables data flow between users' devices and enterprise back-end systems, as well as monitoring usage of mobile apps.
- Supports performance, availability, scalability, and reporting.
- Ensures that security-sensitive data is encrypted locally on device storage.

The following diagram shows the Magnet Mobile App Server components and their relationships.



Magnet Mobile Enterprise Server

Magnet Mobile Enterprise Server (MES) is a server that contains all business logic components in a single deployable JAR file. Each encapsulation of business logic is herein referred to as a Controller. Instances of the Magnet Mobile Enterprise Server are deployed in the cloud, and may contain a blend of off-the-shelf (OTS) and custom controllers that you write for your app.

Custom Controllers

Should you want to create your own custom controllers, you can download the source code for a reference server controller implementation. You can then modify this reference controller implementation to add your own business logic, then upload the modified project for your custom controller back into the Magnet Developer Factory so that this custom controller can be included into a build of the MES instance along with other controllers defined in your project. Custom controllers leverage services such as caching and off-line mode and persistence.

Caching and Off-line Message Delivery

To deal with unreliable connections between mobile devices and MES and to ensure reliable message delivery, Magnet Mobile App Server supports off-line operations that store and cache messages, and forwards information when the network connection is reestablished.

You can invoke these asynchronous services locally on the server or remotely from a mobile client. The invocations from the mobile client can occur asynchronously and reliably (reliability is an option). They are asynchronous in that the invocation is recorded for later playback to the server. They are reliable in that after an invocation has been submitted, it is guaranteed to be run once—and only once from the server perspective.

After entries are processed and results arrive back from the server, a listener that you optionally provide will be notified. Listeners are either transient or durable. A transient listener will be forgotten if the app JVM instance is cycled for any reason. Durable (non-transient) listeners will survive JVM restarts.

Persistence

Persistence is about the storage and retrieval of structured data. Magnet Mobile App Server provides support for entities and mapping entities to the data store. This creates, in effect, a “virtual entity data store” over any combination of MySQL, LDAP, and so forth, that can be used from within the programming interface Magnet provides.



Magnet Mobile App Server uses Java interfaces to define functions, methods, classes, and requests. Magnet Mobile App Server generates implementation from these classes built on annotations declared on interfaces, which allows for relationship, attributes, and so forth. Using user-friendly interfaces on entities, the Magnet Mobile App Server provides these features:

- Queries on entity with query composition
- CRUD (Create, Read, Update, Delete) operations on entities
- Complex queries such as paging and ordering

A developer can choose any technology (JDBC, Hibernate, etc.) to access a data store. However, implementing with Magnet persistence APIs is easier to use, and powerful over time. Every time an entity is used, unstructured data is collected. Over time, recommendations can be derived from that unstructured data. For example, when something is changed on an employee record, the implementation will track when that change is made and by whom. Of course, the recommendations will only be provided if you choose to incorporate those modules into your app. The entire methodology of building apps is modular - allowing you to blend the right combination of modules that make sense for your app.

Transactions

Magnet Mobile App Server can optionally use transactions to maintain the integrity of data.

Service Integration

Service integration can automatically transform Web applications from legacy enterprise application servers to the Magnet Mobile Enterprise Server. It exposes traditional SOAP-based Web Services hosted in the enterprise application servers to REST-based services for mobile access. Using the free on-line Developer Factory, you create a project with specific requirements, and download the generated APIs that abstract SOAP/REST services as controllers. Once generated, you can customize them as you see fit.

Third-Party Service Controllers

The third-party service controllers are sample controllers that allow you to connect to well-known social services, such as Facebook, LinkedIn, or Salesforce. These controllers have built-in support for OAuth, so you can securely retrieve contacts information, and share an update. The source code for these controllers is included in the project generated by the Developer Factory if you choose to include them in your project.



Magnet Mobile App Manager

The Magnet™ Mobile App Manager (MAM) is a run-time server that monitors the service status of Magnet MES and provides app management functionality by way of the Magnet Mobile App Management Console and Mobile App Store.

Mobile App Management Console

The Magnet MAM Console is a Web interface of the Magnet Mobile App Manager that enables IT administrators to centrally manage and administer apps on client devices. Access to the MAM Console is defined by role-based and group association, which is defined in LDAP. Using the MAM Console, IT administrators can:

- Enable employee BYOD.
- Remotely install, remove, upgrade, and track apps.
- Customize an enterprise private-label app store.
- Remotely activate / deactivate user accounts.
- Push required apps to authorized users.
- View logs to monitor events history.

Mobile App Store

The Mobile App Store is an app for installing and managing apps on a mobile device and is installed on iOS and Android devices. The Mobile App Store includes a suite of sample apps that are organized in categories. Via the MAM Console, IT administrators can brand the Mobile App Store with the enterprise private label. The App Store branding kit can be obtained on request.

Deployment

An executable JAR file contains all controllers required for a specific deployment. Instances of the Magnet Mobile Enterprise Server are deployed in the Amazon cloud—a virtual private cloud that is managed by customers.



Developer Factory

The Developer Factory is a free on-line service that provides a secure and private place for you to dynamically construct controller interfaces specific to the back-end services required for use in your mobile apps. The Developer Factory empowers you to

- Create and manage your app projects.
- Generate customized controllers and runtime binaries for your projects.
- Obtain source code for all your customized project assets.
- Upload modified controller and entities source codes for custom app controllers to regenerate executable JAR files.
- Obtain binaries for additional components required to deploy your projects in the Amazon cloud. Automatically create other language packs of your controllers (for example, iOS/XCode).
- Test-drive your app projects by getting access to your own private sandbox environment in the cloud.





3. Getting Started

The Magnet Mobile App Server uses Apache Maven, an open source tool used for building and assembling application.

- Java 1.6 for Android-based applications
- Maven 3.0.3 or later

If you are using a recent version of MacOS, Maven may be already installed. For a free download and installation, visit

<http://maven.apache.org/download.html>

Maven uses an XML file to describe the software project being built, its dependencies on other external modules and components, an Internet-based repository to automatically retrieve off-the shelf (OTS) modules produced by Magnet or other open source contributors as needed, the build order, directories, and required plug-ins. The Project Object Model (POM) file contains everything needed to describe a project. This file is automatically created when creating a Maven project from the Developer Factory.

Setting Up Your Repository Environment

You can set up the repository using one of the following methods:

- create a settings.xml file
- declare in the POM file

Creating a settings.xml File

Place the *settings.xml* file in the **.m2** directory. An example of the settings.xml file is shown.

```
<settings>
  <profiles>
    <profile>
      <id>magnet</id>
      <repositories>
        <repository>
          <id>maven_magnet</id>
          <name>Public Magnet Maven Repository</name>
          <url>http://developer.magnet.com/nexus/content/
groups/public/</url>
        </repository>
      </repositories>
    </profile>
  </profiles>
  <activeProfiles>
    <activeProfile>magnet</activeProfile>
  </activeProfiles>
</settings>
```

Declaring in the POM File

You can declare the repository for your project by adding the following information to your POM file.

```
<project ...="">
  ...
  <repositories>
    ...
    <repository>
      <id>maven_magnet</id>
      <name>Public Magnet Maven Repository</name>
      <url>http://developer.magnet.com/nexus/content/
groups/public/</url>
    </repository>
  </repositories>
  ...
</project>
```



Visit <http://maven.apache.org/> or <http://maven.apache.org/guides/getting-started/> for getting additional information about Maven.



Setting up Your IDE

The project downloaded from the Magnet Developer Factory uses Maven. A non-Maven project is also generated to package the libraries and the service configuration assets in conventional Android project format that can be imported to Eclipse or built using ant directly.

You need the following items installed on your local machine:

- Android SDK
- Eclipse Android Plugin

Refer to the Android documentation for information.

Setting up Your Account in the Magnet Developer Factory

The Developer Factory is a free on-line service that provides a secure and private place for you to dynamically construct controller interfaces specific to the back-end services required for use in your mobile apps and on your server instance(s).

To request a Developer Factory account:

- 1 Launch a browser and enter the following URL:

<http://factory.magnet.com>

The screenshot shows the Magnet Developer Factory login and registration interface. On the left, under the heading "Sign In", there is a message "You must Sign In to access the Developer Factory." Below this are two input fields: "Email Address*" and "Password*". A red "Sign in" button with a lock icon is positioned below the fields, and a red link "Forgot Password?" is located underneath the button. On the right, under the heading "Don't have an account?", there is a message "Request an invitation to start developing now!". Below this is a graphic showing a hand holding a smartphone with various app icons floating around it. To the right of the graphic is a list of bullet points: "Quickly build apps", "iOS™ and Android™ platforms", "Use your own web services", "User pre-defined 3rd party web services", and "Deploy your project to the cloud". Below the graphic and list is a red "Request Invitation" button and a red link "More Information >>". At the bottom right, a note states: "Note: By pressing the 'Request an Invitation' button above, you will be taken to [www.magnet.com](\"http://www.magnet.com\") to request an invitation to the Magnet™ Developer Factory."

- 2 Click **Request Invitation** to start the process.





4. Managing Your Project with Developer Factory

The Developer Factory provides an intuitive environment enabling you to easily build and manage your project. As an owner of a project that you create, you can:

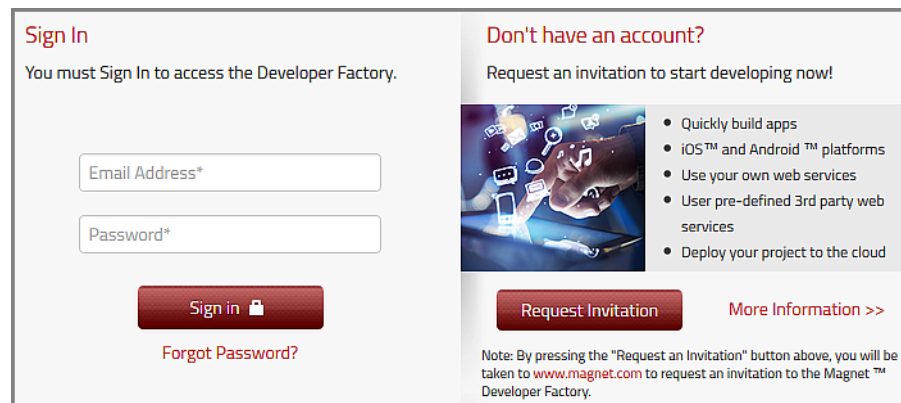
- Create a project (page [14](#))
- Deploy your project for testing or download it for customization (page [21](#))
- Modify your project (page [22](#))
- Upload a customized project (page [24](#))
- Remove your project (page [25](#))
- Invite other developers (page [27](#))

Creating a Project

The Magnet Developer Factory provides wizard that walks you through the creation of a project.

- 1 Launch a browser and enter the following URL:

<http://factory.magnet.com>

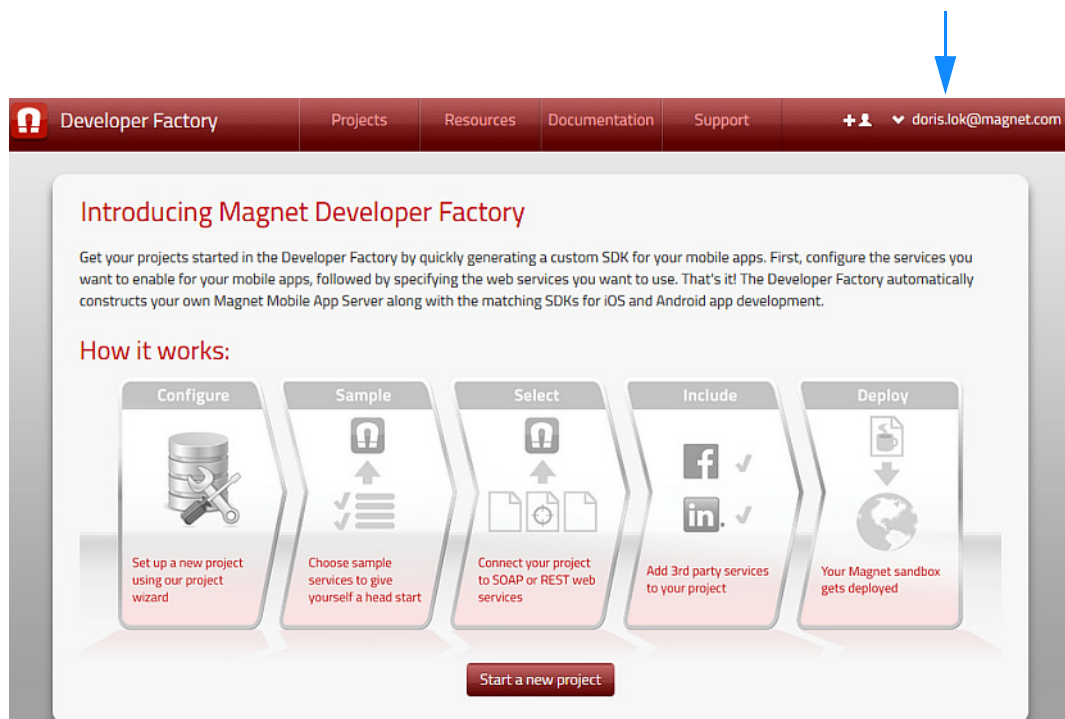


The screenshot shows the login interface of the Magnet Developer Factory. On the left, under the heading "Sign In", there is a message: "You must Sign In to access the Developer Factory." Below this are two input fields: "Email Address*" and "Password*", followed by a "Sign in" button with a lock icon and a "Forgot Password?" link. On the right, under the heading "Don't have an account?", there is a message: "Request an invitation to start developing now!". Below this is a "Request Invitation" button and a "More Information >>" link. A list of features is shown: "Quickly build apps", "iOS™ and Android™ platforms", "Use your own web services", "User pre-defined 3rd party web services", and "Deploy your project to the cloud". A note at the bottom states: "Note: By pressing the 'Request an Invitation' button above, you will be taken to [www.magnet.com](\"http://www.magnet.com\") to request an invitation to the Magnet™ Developer Factory."

- 2 Enter your user credentials information in the appropriate fields:

- 3 Click **Sign in**.

The page with your user name displayed opens, as shown in the following example.



4 Click **Start a new project**.

Create New Project

Project Details

Enter Project Name *

Enter Version *

Enter Project Description *

After completion of the wizard, you will get:

Server

- Magnet services
- Your own web services
- 3rd party services

Client

- Native project
- Native project

Back to Projects List

Next Step

5 Enter the name of your project, version number, and a brief description in the appropriate fields.

6 Click **Next Step**.

The name of the project you created is displayed as shown in the following example.

ABC Project

App Configuration

Third Party Services

Enterprise Services

Sample Services

App Configuration

Encryption Services OFF

Location Data Collection OFF

User Authentication Default User

Google Android Notification Service Not Included

Apple iOS Notification Service Not Included

Email Notification Service Not Included

Back to Projects List

Next Step

7 Click to select the following configuration information:

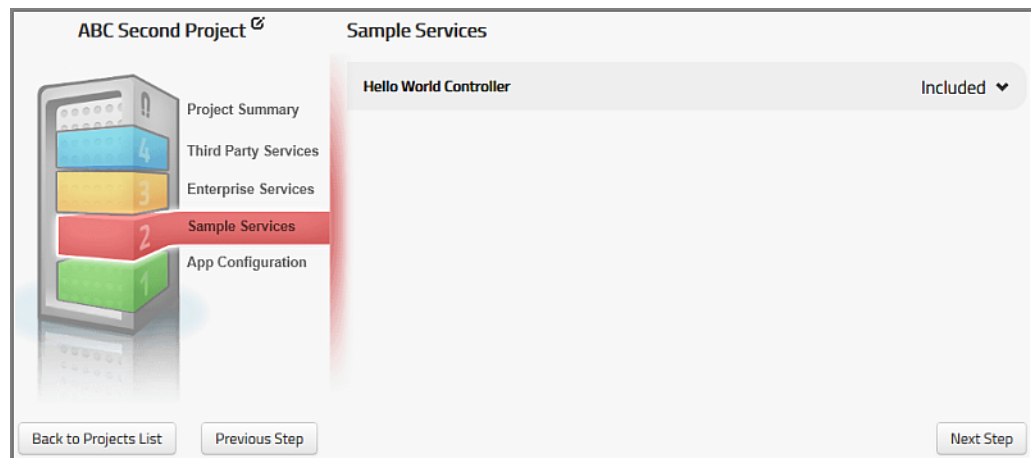
- *Encryption Services*. This is used to encrypt security sensitive information in your configuration as well as on your mobile devices.
- *Location data collection*: This enables geo-tagged coordinates to be transported each time a controller is invoked on your server, allowing your server to know the location of the mobile app user.



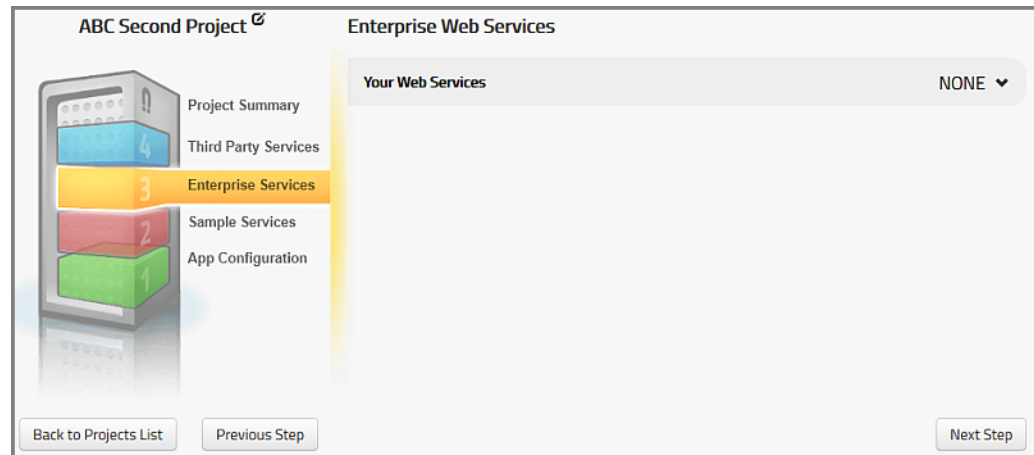
- *User Authentication*: You can select the appropriate data store from where you want to read user credentials to authenticate for accessing server from the mobile clients.
 - Select Default User if you want the Developer Factory to provide a user name and password for you that will get you access to a deployed sandbox.
 - Select the appropriate option if you want to use your own tools and environment to control access to the sandbox. User name and password will be managed by one of these three systems.
- 8 Click **Include** to enable these services so you can utilize those notification capabilities for your apps.
 - Google Android Notification Service
 - Apple iOS Notification Service
 - Email Notification Service
 - 9 Click **Next Step** to select the sample services.
 - 10 Include the Hello World Controller if you want to get a template to start developing your own custom controllers. However, if you do not want to add your own custom controllers, then click **Don't Include**.



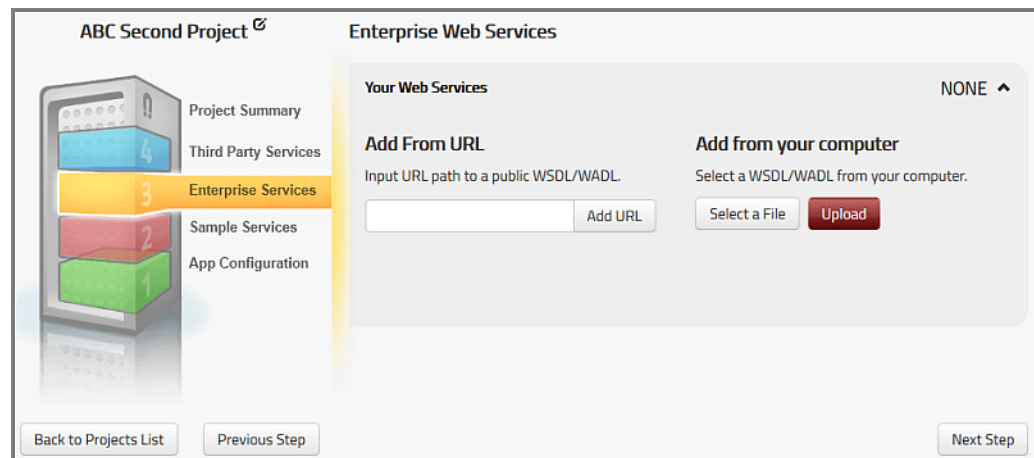
The Hello World Controller provide a simple UI that can be run on an emulator or on a phone and connect to the deployed sandbox (or any server where you may have installed our generated server JAR).



11 Click **Next Step** to add your Web services.



12 Click **NONE** to display the Web Services options.



13 Enter the URL to add your WSDL or WADL file, click **Add URL**.

— Or —

Click **Select a File** to add your WSDL or WADL file from your computer, click **Upload**.

14 Configure security settings for your app to properly communicate with this Web service.



The security configuration window does not appear if security configuration is included in your WSDL file.



The screenshot shows the 'Enterprise Web Services' configuration interface. On the left, a sidebar lists four categories: 'Third Party Services' (4), 'Enterprise Services' (3), 'Sample Services' (2), and 'App Configuration' (1). The 'Enterprise Services' category is selected. The main area is titled 'Your Web Services' and shows '1 Services'. It has two tabs: 'Add From URL' and 'Add from your computer'. The 'Add From URL' tab is active, showing a text input for 'Input URL path to a public WSDL/WADL.' and an 'Add URL' button. Below this, a 'WSDL' section shows the URL 'http://raspi7.magnet.c...' and a warning 'No WS Policy detected. Select a setting below:'. There are four radio button options: 'Basic Auth' (selected), 'UsernameToken', 'None', and 'Specify later'. Each option has a description. At the bottom, there are buttons for 'Back to Projects List', 'Previous Step', and 'Next Step'.

- When Basic Auth is selected, the following window appears:

This screenshot shows the configuration window for 'Basic Auth'. It has the same header as the previous window. The 'Basic Auth' radio button is selected. Below the radio buttons, there are two text input fields for 'ID' and 'Password'. Below these fields are two buttons: 'Timestamp ON' and 'Timestamp OFF'. At the bottom, there is a note: 'You may leave Username and Password blank if you want to specify them later.'

- When UsernameToken is selected, the following window appears:

This screenshot shows the configuration window for 'UsernameToken'. It has the same header as the previous window. The 'UsernameToken' radio button is selected. Below the radio buttons, there are two text input fields for 'ID' and 'Password'. Below these fields are two buttons: 'cleartext' and 'digest'. Below these buttons are two buttons: 'Timestamp ON' and 'Timestamp OFF'. At the bottom, there is a note: 'You may leave ID and Password blank if you want to specify them later.'



15 Click **Next Step** to display the *3rd Party Services* page.

The screenshot shows the '3rd Party Services' configuration page for a project named 'ABC Second Project'. On the left, a sidebar contains a stack of four colored boxes labeled 1 through 4, representing the project setup steps: 1 (green) for App Configuration, 2 (red) for Sample Services, 3 (yellow) for Enterprise Services, and 4 (blue) for Third Party Services. The 'Third Party Services' step is currently selected. The main area displays a list of services: Salesforce, LinkedIn, and Facebook, each with a status of 'Not Included' and a dropdown arrow. At the bottom, there are three buttons: 'Back to Projects List', 'Previous Step', and 'Next Step'.

16 Select the third-party services that you want to include in your application, click **Include**.

This screenshot shows the '3rd Party Services' page after the 'Salesforce' service has been selected for inclusion. The 'Salesforce' service is now marked as 'Included' with an upward arrow. Below the service name, a message states 'Your project is including this third party service.' and there are two buttons: 'Include' (highlighted in red) and 'Don't Include'. Underneath, there is a section titled 'Optional Parameters' containing a 'Credentials' subsection with input fields for 'Security Token' and 'Password'. A note below these fields says: 'Note: If you do not have the information above, please visit: [How to get Salesforce Security Token](#)'. The 'LinkedIn' and 'Facebook' services remain listed below with a 'Not Included' status. A 'Next Step' button is located at the bottom right.

17 Optionally enter the credentials and password (you received those from Connected Apps) in each of the services you include.

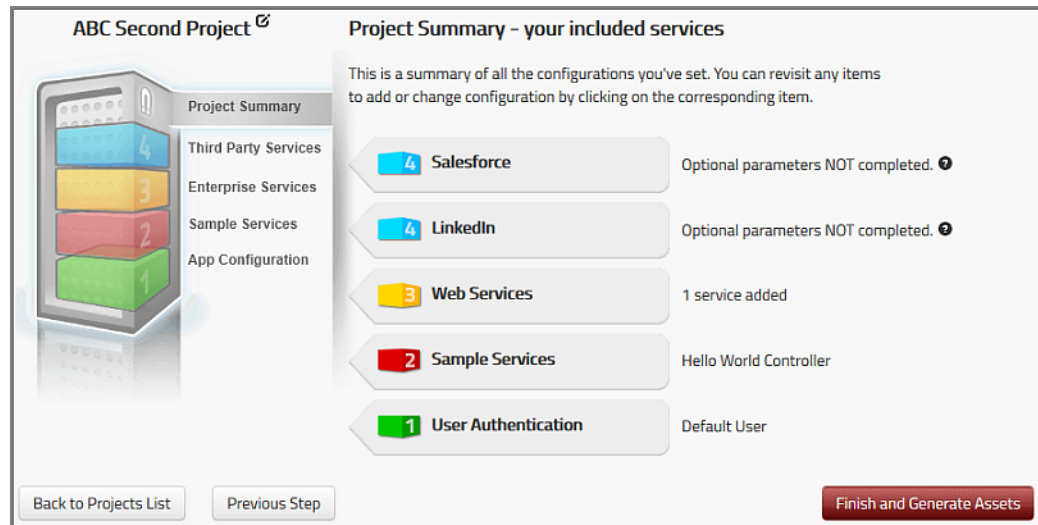


You can configure these settings later in your generated project.



18 Click **Next Step**.

Your project summary is displayed as shown in the following example.

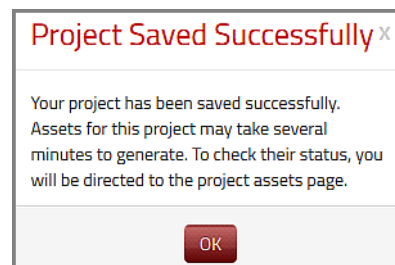


The color-coded buttons shown on the Project Summary side are now interactive. Clicking them takes you directly to the intended page. For example, clicking User Authentication opens the App Configuration step.

19 Review your project.

- To make changes, click **Previous Step** to the step that you need changing.
- Click **Finish and Generate Assets** if no further changes are needed.

A dialog box appears informing of the project status.



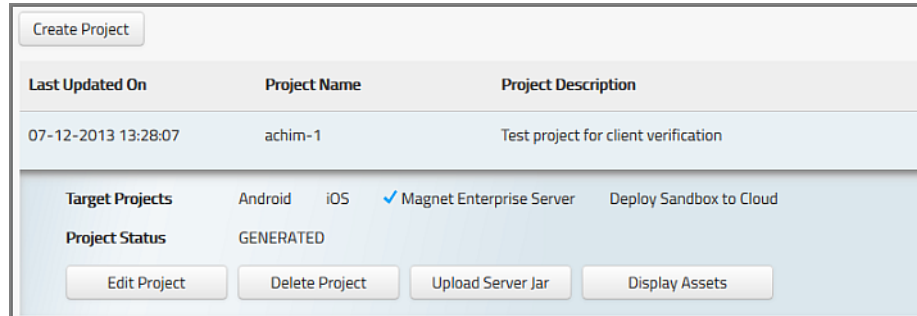
20 Click **OK** to start the asset generation process.



Displaying the Generated Assets

After your project is generated, you can download it to your local machine for customization or deploy it to the sandbox for testing.

- 1 Click the **Projects** tab, and select the intended project.



Last Updated On	Project Name	Project Description
07-12-2013 13:28:07	achim-1	Test project for client verification

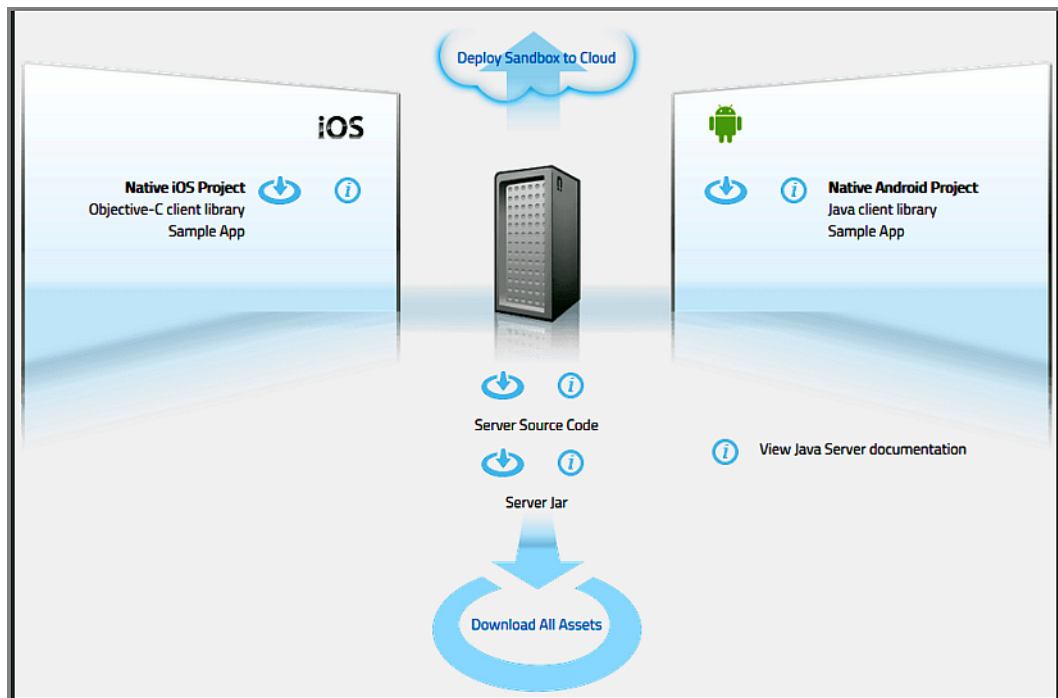
Target Projects	Android	iOS	✓ Magnet Enterprise Server	Deploy Sandbox to Cloud
Project Status	GENERATED			

[Edit Project](#)
[Delete Project](#)
[Upload Server Jar](#)
[Display Assets](#)

- 2 Click **Display Assets**.



You are informed if the project has not been generated. Click **Generate Assets**, then repeat step 2.



- 3 Click **Deploy Sandbox to Cloud**, or select the project you want to download, or click **Download All Assets** to download all assets.

Each server project is compressed into a zip file.



Modifying a Project



Do not change the maven groupId, artifactId, and version of your project if you plan to regenerate the mobile assets out of a modified project.

- 1 Sign in to your Developer Factory account.

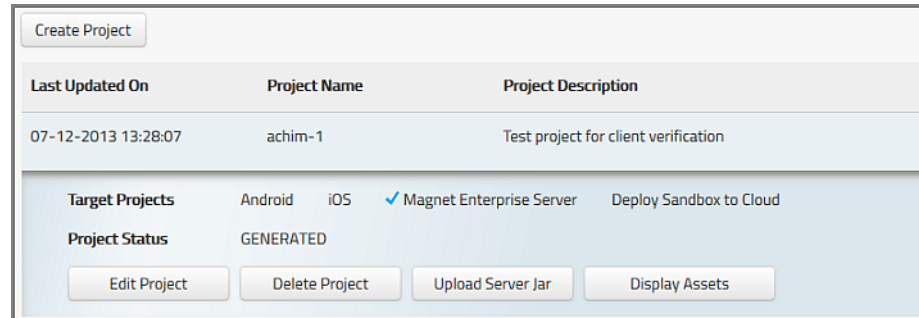


- 2 Click the **Projects** tab to display all projects that you have created, as shown in the following example.

Create Project		
Last Updated On	Project Name	Project Description
07-05-2013 11:02:41	ABC Project	WSDL
07-03-2013 17:21:12	hoa-july-4th-project	Test QA dev factory project
07-03-2013 14:25:31	jim 3	test
07-03-2013 13:43:47	hoa-weblogic-awesomeservice	project to celebrate July 4th



3 Select the project you want to modify.



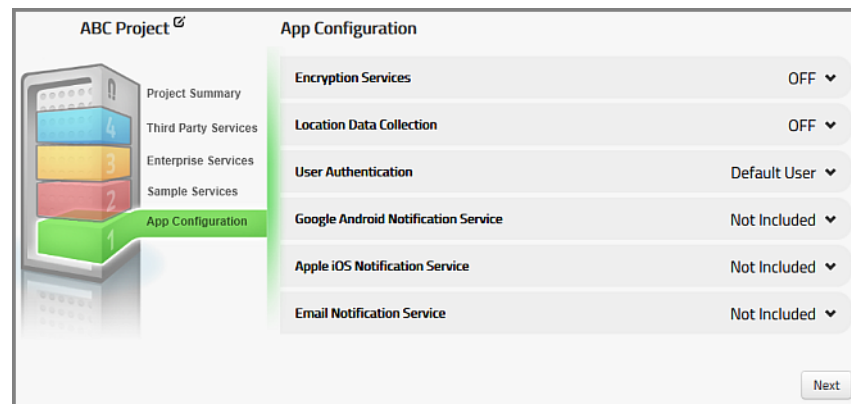
The screenshot shows a web interface for managing projects. At the top is a 'Create Project' button. Below it is a table with three columns: 'Last Updated On', 'Project Name', and 'Project Description'. The table contains one row with the values '07-12-2013 13:28:07', 'achim-1', and 'Test project for client verification'. Below the table, there are sections for 'Target Projects' (listing Android, iOS, and Magnet Enterprise Server with a checkmark) and 'Project Status' (showing GENERATED). At the bottom are four buttons: 'Edit Project', 'Delete Project', 'Upload Server Jar', and 'Display Assets'.

4 Click **Edit Project**.

A page similar to the following example appears.



All color-coded buttons are now interactive. Clicking them takes you directly to the intended page.



The screenshot shows the 'ABC Project' configuration page. On the left is a sidebar with a stack of four color-coded layers: 'Project Summary' (blue), 'Third Party Services' (orange), 'Enterprise Services' (red), and 'Sample Services' (green). The 'App Configuration' layer is highlighted in green and labeled with a '1'. The main area on the right is titled 'App Configuration' and contains several settings: 'Encryption Services' (OFF), 'Location Data Collection' (OFF), 'User Authentication' (Default User), 'Google Android Notification Service' (Not Included), 'Apple iOS Notification Service' (Not Included), and 'Email Notification Service' (Not Included). A 'Next' button is at the bottom right.

5 Click the intended color-coded services layers and modify any information (refer to *Creating a Project* for detailed description).



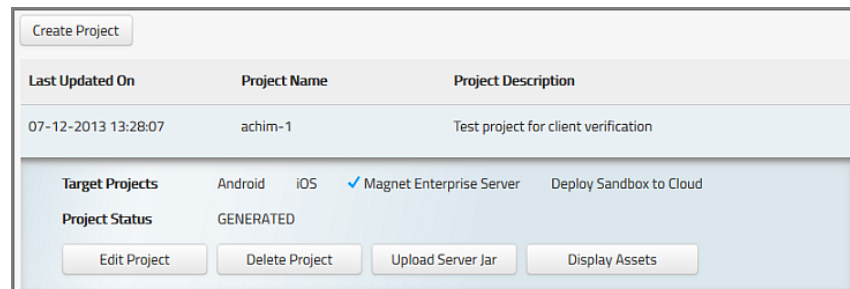
Uploading a Project

When your project customization is complete, you need to upload the project from your local machine to the Developer Factory, and regenerate the assets.

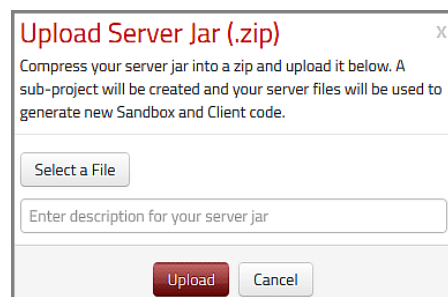
- 1 Sign in to your Developer Factory account.



- 2 Click the **Projects** tab to display all projects that you have created.
- 3 Select the project you want to upload.



- 4 Click **Upload Server Jar**.



- 5 Click **Select a File**, and navigate to the *aws-Server/target* directory where the server JAR zip is stored.



Refer to *Chapter 9, Compiling Your Project*, for the directory structure of the project.

- 6 Enter an optional description for the server JAR.
- 7 Click **Upload**.

Removing a Project

When you close your Developer Factory account, all projects that you owned will be automatically removed, and you will be removed from all projects to which you had been invited.

- 1 Sign in to your Developer Factory account.

The screenshot shows the Magnet Developer Factory web interface. At the top is a navigation bar with links for Projects, Resources, Documentation, and Support, along with a user profile icon and email address (doris.lok@magnet.com). The main content area is titled "Introducing Magnet Developer Factory" and describes the process of generating a custom SDK for mobile apps. Below this, a section titled "How it works:" displays a five-step wizard process:

- Configure:** Set up a new project using our project wizard (icon: database and wrench).
- Sample:** Choose sample services to give yourself a head start (icon: document with checkmark).
- Select:** Connect your project to SOAP or REST web services (icon: document with gear).
- Include:** Add 3rd party services to your project (icon: Facebook and LinkedIn logos with checkmarks).
- Deploy:** Your Magnet sandbox gets deployed (icon: globe with download arrow).

At the bottom of the wizard is a button labeled "Start a new project".



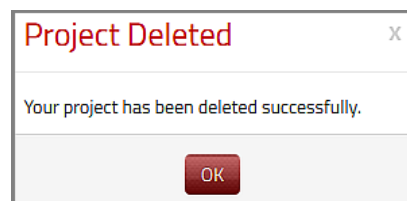
- Click the **Projects** tab to display all projects that you have created, as shown in the following example.

Create Project		
Last Updated On	Project Name	Project Description
07-05-2013 11:02:41	ABC Project	WSDL
07-03-2013 17:21:12	hoa-july-4th-project	Test QA dev factory project
07-03-2013 14:25:31	jim 3	test
07-03-2013 13:43:47	hoa-weblogic-awesomeservice	project to celebrate July 4th

- Select the project you want to remove.

Create Project		
Last Updated On	Project Name	Project Description
07-12-2013 13:28:07	achim-1	Test project for client verification
<div>Target Projects</div> <div>Android iOS <input checked="" type="checkbox"/> Magnet Enterprise Server Deploy Sandbox to Cloud</div> <div>Project Status</div> <div>GENERATED</div> <div>Edit Project Delete Project Upload Server Jar Display Assets</div>		

- Click **Delete Project**.
A dialog box appears asking for confirmation.
- Click **Continue**.
A dialog box appears showing the operation status.



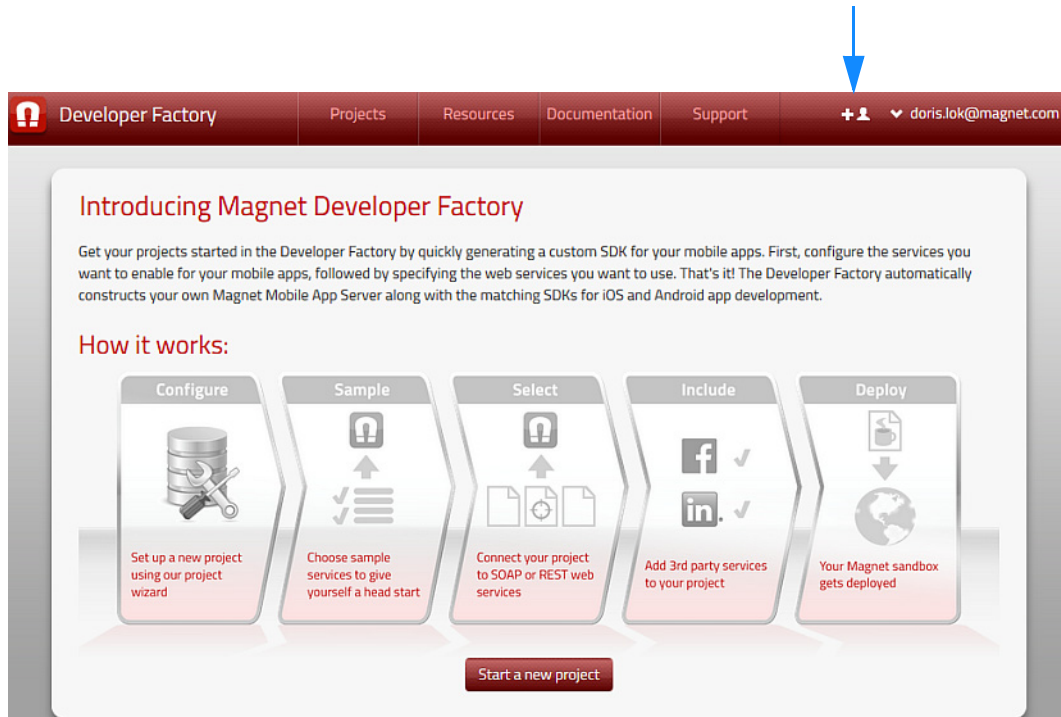
- Click **OK**.
The project is immediately removed from the project page.



Inviting Other Developers

You can invite other developers to the Magnet Developer Factory to create their own accounts. If your invitees are not registered with Developer Factory, they need to first register for an account.

- 1 Sign in to your Developer Factory account.



- 2 Click **+ person** to display the invitation, as shown in the following:

The screenshot shows a dialog box titled 'Invite Others' with a close button (X) in the top right corner. The dialog contains the following elements:

- A text input field labeled 'Enter Email Address'.
- A section titled 'Message to include' with a text area containing the text: 'I found this great new tool to quickly create mobile apps for me.'
- Two buttons at the bottom: 'Send Invitation' and 'Close'.



3 Enter the e-mail address of the person you want to invite, and optionally delete the existing message from the **Message to include** box, and enter your own message that you want to include with your e-mail.

4 Click **Send Invitation**.

You are informed of the operation status. Your recipient will receive an e-mail notification that includes a link to register an account.



After you have downloaded your project from the Magnet Developer Factory to your local environment, you can modify your project, build it, then upload the new project to the Developer Factory for new assets generation.

The project contains files to support both B2C and B2E apps. By default, connection property values are set for B2C apps to target Magnet Enterprise Server running in the sandbox. The *AndroidManifest.xml* file is generated based on the package name and options selected from the Magnet Developer Factory.

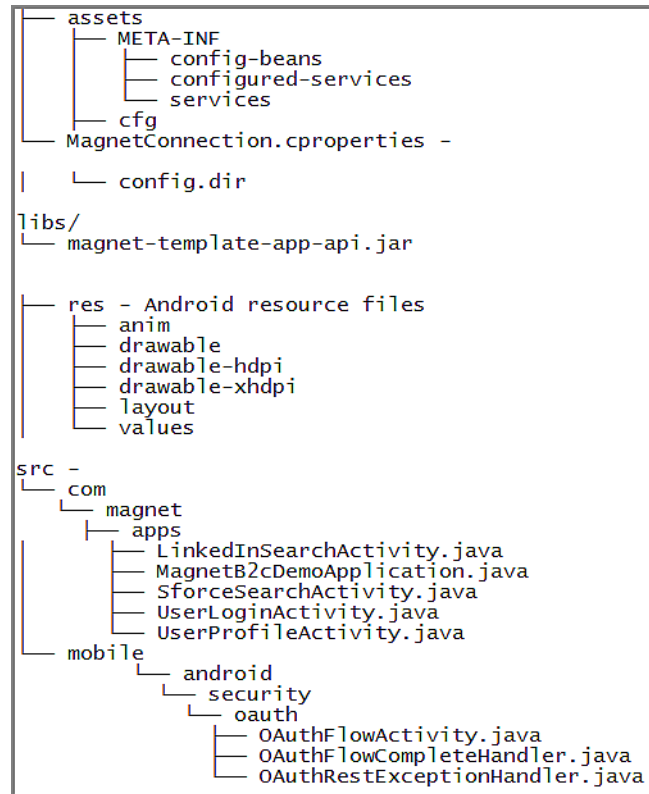


B2C apps can be deployed without use of Mobile Server.

Understanding the Project Structure

An Android project is structured with directories as shown in the following diagram. Certain directories (such as META-INF) contain files that should not be modified while others can be changed.

- **META-INF** contains required kernel configuration files. **Do not modify.**
- **MagnetConnection.cproperties** contains connection properties to the Magnet Enterprise Server instance running in the sandbox of Dev Factory. You can change the URL pointing to a different server.
- **config.dir** contains required kernel service configuration files. **Do not modify.**
- **magnet-template-app-api.jar** contains dependent Magnet specific library JAR generated from Magnet Developer Factory depending on the options selected by you. This is tightly coupled to the configuration files stored in the *assets* directory.



- **src** contains Java source files that are generated based on your input in the Magnet Developer Factory.
 - *LinkedInSearchActivity.java* shows how to connect to LinkedIn and perform a contact search using the LinkedIn controller.
 - *MagnetB2cDemoApplication.java* is the top level Android Application.
 - *SforceSearchActivity.java* shows how to connect to Salesforce and perform a contact search using the Salesforce controller.
 - *UserLoginActivity.java* shows how to login to the Magnet Enterprise Server. This is the main activity that is invoked when application starts. It uses connection properties from `cfg/MagnetConnection.cproperties` to make the connection.
 - *UserProfileActivity.java* shows how to get profile for the currently logged in user using “LoggedInUserController” controller.
- **oauth** contains UI and notifications for OAuth authentication login redirects. This is used by *LinkedInSearchActivity* and *SForceSearchActivity* if *OAuthRestException* occurs when the controller method is invoked.



Building the Project with Eclipse

Make sure that the Android SDK and Eclipse Android Plugin are installed. Refer to Android documentation for information.

- 1 Unzip the project zip file, and note the name of the directory.
- 2 Create a new project from Eclipse (File->New Project->Android->Android Project from Existing Code).
- 3 Browse to the unzipped project directory, and select it as the “Root Directory.”
A new Android project, **UserLoginActivity**, is created.
- 4 Build the Android APK (Project->Clean)
– Or –
Right-click on the project name, and select **Build Project**.
- 5 Deploy the built APK to an Android device or emulator to run the app.
When the app first starts, it will prompt for login info.





6. Defining Application Programming Interfaces

This chapter provides API methods for the following services:

- MagnetMobileClient (page [34](#))
- Enum MagnetMobileClient.Connection State (page [42](#))
- Connection Service (page [44](#))
- Connection Service Listener (page [46](#))
- EmbeddedMobileManager (page [49](#))
- MessageHandler (page [53](#))
- Messenger (page [58](#))
- Messenger.DeliveryType (page [65](#))
- MagnetActivity (page [67](#))
- MagnetAndroidService (page [69](#))
- MagnetApplication (page [70](#))
- AndroidSettingsConstants (page [71](#))

MagnetMobileClient

MagnetMobileClient is a class that helps to initialize kernel and basic connections to the Magnet Mobile Server or Magnet Mobile Enterprise Server.

```
public class MagnetMobileClient
extends Object
```

General Magnet app settings are stored in Android shared preferences and there are two ways to get an instance:

- 1 Via Android API **Context.getSharedPreferences(String name, int mode)**,

where

name=**com.magnet.mobile.android.settings.MAGNET_SHARED_PREFERENCE_FILE_NAME**

- 2 Via **@Inject** construct.

For example:

```
@Inject

@Named(com.magnet.mobile.android.AndroidSharedPreferencesProvider.DEFAULT_NAME)
private Reference sharedPrefs;
```

The following table lists and describes *MagnetMobileClient* methods and a page reference to each method.

Connection Methods	Description	Page
connect()	Connect to the Magnet Mobile Server via mobile server ibinder interface using single-sign-on	page 35
connect(String username, String password, String authority)	Connect to the Magnet Mobile Server using its own credential.	page 36
connect(com.magnet.mobile.android.ConnectionPreferences preferences)	Connect to the Magnet Mobile Server using specified preferences.	page 36
disconnect()	Disconnect from Magnet Mobile Server.	page 37
getConnectionPreferences()	Return the instance of connection preferences.	page 37
getConnectionService()	Return the instance of ConnectionService.	page 38
getConnectionState()	Return the current connection state.	page 38
getService()	Return a registered service by its class.	page 39



Connection Methods	Description	Page
<code>injectNow()</code>	Inject data to the container.	page 39
<code>obtainPreferences()</code>	Return an instance of connection preferences.	page 40
<code>registerConnectionListener()</code>	Registers connection listener after the kernel has been started.	page 40
<code>startKernel()</code>	Start the kernel and return services, such as Service Finder, Connection Preferences, and Connection Service.	page 41
<code>stopKernel()</code>	Stop the kernel and releases the services that were returned when the kernel was started.	page 41
<code>unregisterConnectionListener()</code>	Unregister connection listener.	page 42

connect()

This method is used for connection with single-sign-on. This method connects to the Magnet Mobile Server by way of the mobile server ibinder interface using single-sign-on. If there is an established connection, calling this method will disconnect the connection and establish a new one.

To check the connection state, you can invoke *getConnectionState()* on page **38** before invoking the `connect()` method.

Call `ConnectionServiceListener.onConnectionEvent(ConnectionService, com.magnet.connection.ConnectionServiceListener.Event)` for the connection result.

```
public boolean connect()
    throws com.magnet.common.PlatformRuntimeException
```

Return

true

Exception

`PlatformRuntimeException` indicates that the kernel has not started.



connect(String username, String password, String authority)

This method is used for connection with its own credential. Credentials are saved in an encrypted form to a persistent storage whether or not the connection is successful.

To check the connection state, you can invoke *getConnectionState()* on page 38 before invoking the connect() method.

Implement your own ConnectionServiceListener @Singleton service in order to subscribe to connection events with the server.

```
public boolean connect(String username,
    String password,
    String authority)
    throws com.magnet.common.PlatformRuntimeException
```

Return

true

Exception

PlatformRuntimeException indicates that the kernel has not started.

connect(com.magnet.mobile.android.ConnectionPreferences preferences)

This method is used for connection using specified preferences. The connection parameters and credential are encrypted and persisted in a storage whether or not the connection is success. If preferences is null, the call is treated the same as the *connect()* method. You can retrieve an instance of connection preferences by calling the *obtainPreferences()* method (page 40).

To check the connection state, you can invoke *getConnectionState()* on page 38 before invoking the connect() method.

Call ConnectionServiceListener.onConnectionEvent(ConnectionService, com.magnet.connection.ConnectionServiceListener.Event) for the connection result.

```
public boolean connect(com.magnet.mobile.android.ConnectionPreferences
    preferences)
    throws com.magnet.common.PlatformRuntimeException
```



Parameters

preferences indicates an instance of `ConnectionPreferences`. If null, this method is treated the same as the `connect()` method.

Return

true

Exception

- `PlatformRuntimeException` indicates that the kernel has not started.
- `IllegalArgumentException` indicates that the preferences object is not part of `obtainPreferences()`.

disconnect()

Disconnect from the Magnet Mobile Server. Note that network connection can be active to the server without being authenticated. Thus, this method should be invoked to disconnect the caller from the server at the network level whether or not the caller was able to successfully authenticate.

```
public boolean disconnect()
```

Parameters

None

Return

- true indicates the Magnet Mobile Server is disconnecting.
- false indicates the Magnet Mobile Server is not connected.

Exception

None

getConnectionPreferences()

Return the global instance of connection preference used by the Connection Service.

```
public com.magnet.mobile.android.ConnectionPreferences getConnectionPreferences()
    throws com.magnet.common.PlatformRuntimeException
```

Parameters

None

Return

The instance of `ConnectionPreferences`.

Exception

`PlatformRuntimeException` indicates that the kernel has not started.



getConnectionService()

Return the global instance of ConnectionService.

```
public com.magnet.connection.ConnectionService getConnectionService()  
    throws com.magnet.common.PlatformRuntimeException
```

Parameters

None

Return

The instance of ConnectionService, or null if ConnectionService is not available.

Exception

PlatformRuntimeException indicates that the kernel has not started.

getConnectionState()

Return the current connection state.

```
public MagnetMobileClient.ConnectionState getConnectionState()
```

Parameters

None

Return

A ConnectionState value. Refer to *Enum MagnetMobileClient.Connection State* on page 42 for enum values.

Exception

None



getService()

Return a registered service by its class.

```
public <T> T getService(Class<T> clazz)
```

Parameters

clazz indicates the service class.

Return

The instance of class or null if the class is not available.

Exception

PlatformRuntimeException indicates that the kernel has not started.

injectNow()

Inject data to a container. A container is any object (for example, Activity, Application, and so forth) that contains field members to be injected.

The *startKernel()* on page 41 must be called before this method; otherwise, it will be no operation.

```
public void injectNow(Object container)
```

Parameters

container indicates the container object.

Return

None

Exception

None



obtainPreferences()

Return an instance of connection preferences to be used for call to `connect(ConnectionPreferences)`.

```
public static com.magnet.mobile.android.ConnectionPreferences obtainPreferences()
```

Parameters

None

Return

An instance of connection preferences.

Exception

None

registerConnectionListener()

Register connection listener after the kernel has been started.

```
public void registerConnectionListener  
(com.magnet.connection.ConnectionServiceListener listener)  
throws com.magnet.common.PlatformRuntimeException
```

Parameters

listener indicates implementation of connection listener interface, *ConnectionServiceListener*.

Return

None

Exception

PlatformRuntimeException indicates that the kernel has not started.



startKernel()

Start the kernel and initialize the global Service Finder, Connection Preferences, and Connection Service services.

```
public void startKernel(Object object,  
                        android.os.Bundle bundle)
```

Parameters

- object
- bundle indicates Android Bundle, such as one from Activity.onCreate(Bundle) or null if not applicable.

Return

N/A

Exception

None

stopKernel()

Stop the kernel and release the services that were returned by the *startKernel()* method.

```
public void stopKernel()
```

Parameters

None

Return

N/A

Exception

None



unregisterConnectionListener()

Unregister connection listener.

```
public void unregisterConnectionListener()
```

Parameters

None

Return

N/A

Exception

None

Enum MagnetMobileClient.Connection State

```
public static enum MagnetMobileClient.ConnectionState  
extends Enum<MagnetMobileClient.ConnectionState>
```

The following table lists and describes these methods and a page reference to each of the method.

Methods	Description	Page
valueOf()	Return the enum constant of this type with the specified name.	page 43
values()	Return an array containing the constants of this enum type in the order they are declared.	page 44



values

Return an array containing the constants of this enum type in the order they are declared.

```
public static MagnetMobileClient.ConnectionState[] values()
```

This method may be used to iterate over the constants as follows:

```
for (MagnetMobileClient.ConnectionState c :  
MagnetMobileClient.ConnectionState.values())  
    System.out.println(c);
```

Parameters

None

Return

An array containing the constants of this enum type in the order they are declared. The enum type includes:

- CONNECTED
- CONNECTING
- NOT_CONNECTED

Exception

None



valueOf

Returns the enum constant of this type with the name you specified.



Be sure that the name you entered matches an enum identifier exactly. No whitespace characters are allowed.

```
public static MagnetMobileClient.ConnectionState valueOf(String name)
```

Parameter

- *name* is the name of the enum constant to be returned.

Return

The enum constant with the specified name.

Exception

- `NullPointerException` indicates that the argument is null.
- `IllegalArgumentException` indicates that the enum name you specified does not exist.

ConnectionService

The *ConnectionService* interface is a logical abstraction for a connection-oriented service. This interface includes API methods as listed in the following table.

```
@Dynamic
@Contract
public interface ConnectionService
```

Methods	Description	Page
<code>getBaseUri()</code>	The base URI used for this connection.	page 45
<code>isAuthenticated()</code>	Return true if the connection to the server is established with a user that has been authenticated.	page 45
<code>isConnected()</code>	Return true if the connection to the server is established.	page 45
<code>isConnecting()</code>	Return true if the connection to the server is being established.	page 45



getBaseUri()

Return the base URI used for this connection.

```
java.lang.String getBaseUri()
```

isAuthenticated()

Returns true if the connection to the server is established with a user that has been authenticated.

```
boolean isAuthenticated()
```

isConnected()

Returns true if the connection to the server is established.

```
boolean isConnected()
```

isConnecting()

Returns true if the connection to the server is being established.

```
boolean isConnecting()
```



ConnectionServiceListener

The *ConnectionServiceListener* allows your application to be notified upon various connection events. Note that these events will come in synchronously from the perspective of the underlying connection machinery.

```
@Contract
public interface ConnectionServiceListener
```

The *ConnectionServiceListener* interface has one method: `onConnectionEvent()`.

onConnectionEvent()

These events are triggered on the thread that establishes the connection.

```
boolean onConnectionEvent(ConnectionService ocs,
                          ConnectionServiceListener.Event evt)
```

Return

- true indicates that notifications is continued.
- false indicates that notifications to additional listeners are aborted.

Enum Constant

The following table lists and describes Enum constant and a page reference to each of the method.

Enum Constant	Description	Page
AUTHENTICATED	The credentials are valid.	page 47
BAD_CREDENTIALS	The credentials are invalid.	page 47
BAD_VERSION	The connection is invalid due to a mismatch version between client and server.	page 47
CONNECTED	The connection is established.	page 47
CONNECTING	The connection is being attempted.	page 47
DISCONNECTED	The connection is disconnected.	page 48
UNAUTHENTICATED	The credentials are either invalid or no login was attempted.	page 48



AUTHENTICATED

The credentials are valid.

```
public static final ConnectionServiceListener.Event AUTHENTICATED
```

BAD_CREDENTIALS

The credentials are invalid, the next event will be DISCONNECTED.

```
public static final ConnectionServiceListener.Event BAD_CREDENTIALS
```

BAD_VERSION

The connection is invalid due to a mismatch version between client and server.

The extraStatusLine will optionally contain a semi-colon delimited result containing the server version followed by the download URL for the mobile app given the platform (for example, “2.0.1;[someDownloadUrlFor2-0-1]”).

```
public static final ConnectionServiceListener.Event BAD_VERSION
```

CONNECTED

The connection is established. The next possible event state is DISCONNECTED.

```
public static final ConnectionServiceListener.Event CONNECTED
```

CONNECTING

The connection is being attempted.

```
public static final ConnectionServiceListener.Event CONNECTING
```



DISCONNECTED

The connection is disconnected. The next possible event state is CONNECTING.

```
public static final ConnectionServiceListener.Event DISCONNECTED
```

UNAUTHENTICATED

The credentials are either invalid or no login was attempted.

```
public static final ConnectionServiceListener.Event UNAUTHENTICATED
```



EmbeddedMobileManager

EmbeddedMobileManager is a class that provides access to Magnet Mobile Server Android Service. The current release supports integration of mobile messaging via GCM (Google Cloud Messaging) between client applications and Magnet servers.

```
public class EmbeddedMobileManager
extends java.lang.Object
```

An application sample code is shown below.

```
...
EmbeddedMobileManager emm = EmbeddedMobileManager.getInstance(context);
User currentUser;
...
if (emm.isReady()) {
    // Initialize mobile messenger for GCM communication between Android apps
    // and Magnet server

    Messenger messenger = mm.getMessenger(Messenger.DeliveryType.gcm);

    messenger.initUser(currentUser);
    messenger.registerMessageHandler(msgListener);

    messenger.activateClientDelivery(Messenger.DeliveryType.gcm, senderId);
}
}
...
EmbeddedMobileManager.removeInstance(context);
```



The following table lists and describes methods included in *EmbeddedMobileManager* and a page reference to each of the method.

Methods	Description	Page
<code>getInstance()</code>	Return the instance of the manager that starts the local embedded Mobile Server automatically.	page 50
<code>getMessenger()</code>	Return an instance of Messenger for mobile notifications.	page 51
<code>isFeatureAvailable()</code>	Check feature availability.	page 51
<code>isReady()</code>	Return whether local Mobile Server is ready and bound to client application This is set to true when local binding is finished.	page 52
<code>removeInstance()</code>	Remove the instance for this context.	page 52

getInstance()

Return the instance of the manager that starts the local embedded Mobile Server automatically.

To avoid exceptions, invoke ***isReady()*** to verify that the Mobile Server service is connected before accessing this method

When this instance is no longer needed, invoke ***removeInstance()*** to remove this instance to unbind the client application from Mobile Server Android service.

```
public static EmbeddedMobileManager getInstance(android.content.Context context)
```

Parameters

context indicates the application context.

Return

An instance of `EmbeddedMobileManager` for the specified application context.

Exception

None



getMessenger()

Return an instance of Messenger for mobile notifications

```
public Messenger getMessenger(Messenger.DeliveryType type)
```

Parameters

type indicates the delivery type. The current release supports only GCM.

Example

```
public Messenger getMessenger(Messenger.DeliveryType GCM)
```

Return

An instance of Messenger for the specified delivery type.

Exception

None

isFeatureAvailable()

Check the availability of a specific feature.

```
public static boolean isFeatureAvailable(android.content.Context context,  
                                         java.lang.String featureName)
```

Parameters

- *context* indicates the application context to be returned.
- *featureName* indicates the name of the feature to be returned.

Example

```
boolean hasMobileServer =  
EmbeddedMobileManager.isFeatureAvailable(EmbeddedMobileManager.FEAT_ENT_  
MOBILE_SERVER)
```

Return

true if the feature is available (whether or not the Mobile Server is installed on the device).

Exception

None



isReady()

Return true when the local Mobile Server is ready and bound to the client application.



Client apps must wait in a separate thread (not MAIN) if it wants to wait for ready.

```
public boolean isReady()
```

Parameters

None

Return

true when the local binding is finished.

Exception

None

removeInstance()

Remove the instance for this context. When it is removed, the client application is unbound from the local mobile server service.

```
public static void removeInstance(android.content.Context context)
```

Parameters

context indicates the application context.

Example

```
EmbeddedMobileManager.removeInstance(context);
```

Return

None

Exception

None



MessageHandler

This class defines the callback interface for message push notifications service for client registration and messages received from the server.

The *MessageHandler* interface includes callback methods for managing and delivering push notifications from the Magnet server service.

```
@Contract  
public interface MessageHandler
```

An application sample code is shown below.



```

MessageHandler msgListener = new MessageHandler() {

    @Override
    public void onClientRegistered() {
        mGcmRegistered = true;
        Log.d(TAG, "onClientRegistered");
    }

    @Override
    public void onClientUnRegistered() {
        mGcmRegistered = false;
        Log.d(TAG, "onClientUnRegistered");
    }

    @Override
    public void onServerActivated() {
        mActivated = true;
        Log.d(TAG, "onActivated");
    }

    @Override
    public void onServerCancelled() {
        mActivated = false;
        Log.d(TAG, "onCancelled");
    }

    @Override
    public void onMessageReceived(Object message) {
        Log.d(TAG, "onMessageReceived");
        if (message instanceof Bundle) {
            Bundle bundle = (Bundle) message;
            Log.d(TAG, "bundle=" + bundle.toString());
            doNotify(message);
        } else {
            Log.w(TAG, "not handling message class:" + message.getClass());
        }
    }

    @Override
    public void onError(int errorCode) {
        Log.d(TAG, "onError; errorCode=" + errorCode);
    }
};

```



The following table lists and describes methods included in *MessageHandler* and a page reference to each of the method.

Methods	Description	Page
<code>onClientRegistered()</code>	Client is registered with the primary messenger delivery service, such as GCM.	page 55
<code>onClientUnRegistered()</code>	Client is unregister with the primary messenger delivery service, such as GCM.	page 55
<code>onError()</code>	Error message is received from the messenger delivery service.	page 55
<code>onMessageReceived()</code>	Message is received from the messenger delivery service.	page 55
<code>onServerActivated()</code>	Push notification is activated between the client and the Magnet server.	page 55
<code>onServerCancelled()</code>	Push notification is deactivated between the client and the Magnet server.	page 55

onClientRegistered()

Client is registered with the primary messenger delivery service, such as GCM.

```
void onClientRegistered()
```

Parameters

None

Return

None

Exception

None



onClientUnRegistered()

Client is unregister with the primary messenger delivery service, such as GCM.

```
void onClientUnRegistered()
```

Parameters

None

Return

None

Exception

None

onError()

Error message is received from the messenger delivery service.

```
void onError(int errorCode)
```

Parameters

errorCode



Refer to the Google GCM client API doc for error code.

Return

None

Exception

None



onMessageReceived()

A message is received from the messenger delivery service.

```
void onMessageReceived(java.lang.Object message)
```

Parameters

message indicates the Message object. On Android, it is the instance of Bundle.

Return

None

Exception

None

onServerActivated()

Activate message notification with the Mobile Server.

```
void onServerActivated()
```

Parameters

None

Return

None

Exception

None

onServerCancelled()

Cancel message notification with the server.

```
void onServerCancelled()
```

Parameters

None

Return

None

Exception

None



Messenger

This is an interface to use the Magnet Messenger feature for push notifications from the Magnet server to client apps.

```
@Contract
public interface Messenger
```

To use the push notifications feature, the client app should do the following:

- Register callback for receiving activation result by calling the *registerMessageHandler()* method.
- Activate the service by calling the *activateClientDelivery()* method.

When push notification is activated successfully, the client app can receive messages in two ways:

- Call the *registerMessageHandler()* method to register callback for receiving messages.
- Use Android Broadcast Receiver and listen for intent, ACTION_MSG_RECEIVED. This requires permission, PERMISSION_RECEIVE, which is automatically included when generating an Android project by the Magnet Developer Factory.

In addition, because Google Cloud Messaging Service (GCM) is the primary delivery mechanism for Android, the following permission must be declared in client app's AndroidManifest.xml in this form:

```
<permission android:name="<app-package>.permission.C2D_MESSAGE"
android:protectionLevel="signature"/>
<uses-permission android:name="<app-package>.permission.C2D_MESSAGE"/>
```

For example:

```
<permission android:name="com.magnet.demo.message.permission.C2D_MESSAGE"
android:protectionLevel="signature"/>
<uses-permission android:name="com.magnet.demo.message.permission.C2D_MESSAGE"/>
```



An application sample code is shown below.

```
...
EmbeddedMobileManager emm = EmbeddedMobileManager.getInstance(context);
...
if (emm.isReady()) {
    // Initialize mobile messenger for GCM communication between Android
    apps and Magnet server

    // Get an instance of Messenger
    Messenger messenger = mm.getMessenger(Messenger.DeliveryType.gcm);
    // Register callback for activation completion and receiving messages
    messenger.registerMessageHandler(msgListener);
    if (!messenger.isNotificationActivated()) {
        // Activate push notification
        messenger.activateClientDelivery(Messenger.DeliveryType.gcm,
senderId);
    }
}
```

The following table lists and describes methods included in the *Messenger* interface and a page reference to each of the method.

Methods	Description	Page
activateClientDelivery()	Register caller client app for notification delivery from specified sender This is an asynchronous method.	page 60
activateServerNotification()	This API is used internally and should not be called directly by the client app.	page 60
cancelClientDelivery()	Cancel notification delivery to the caller app from messenger service.	page 61
cancelServerNotification()	This API is used internally and should not be called directly by the client app.	page 61
getClientDeliveryToken()	Return the registered client token for notification delivery.	page 62
getMessageHandlers()	Return a list of message handlers.	page 62
getServerNotificationActivated()	Return notification whether or not server notification is activated.	page 62
isDeliveryTypeSupported()	Return if specified delivery type is supported by underlying implementation.	page 63
isNotificationActivated()	Return true if server notification is activated to this app.	page 63



Methods	Description	Page
onMessageReceived()	Invoked by messenger receiver service when message is received from underlying message service, such as GCM.	page 63
registerMessageHandler()	Register message handler in the current process.	page 64
removeMessageHandler()	Remove message handler from the current process.	page 64

activateClientDelivery()

Register caller client app for notification delivery from a specified sender. This is an asynchronous method.

```
boolean activateClientDelivery(Messenger.DeliveryType type,
                             java.lang.String senderId)
```

Parameters

- *type* indicates the messenger delivery type (refer to [Enum Messenger.DeliveryType](#) on page [65](#)).
- *senderId* indicates the identifier of the sender server, such as GCM.

Return

- true indicates that registration has started.
- false indicates that the delivery type is not supported.

activateServerNotification()

Activate notification from the server asynchronously, assuming there is a valid connection to the server and it is connected.

This method is called automatically by the mobile server library after the [activateClientDelivery\(\)](#) method is successful and confirmed by the messaging service.



This API is used internally and should not be called directly by the client app.

```
boolean activateServerNotification(Messenger.DeliveryType type,
                                 java.lang.String receiverId,
                                 java.lang.String packageName)
```



Parameters

- *type* indicates the messenger delivery type (refer to *Enum Messenger.DeliveryType* on page 65).
- *receiverId* indicates the identifier of the receiver, such as GCM.
- *packageName* indicates the package name of the client receiver.

Return

- true indicates that activation has started.
- false indicates an error, such as no network connection.

cancelClientDelivery()

Cancel notification delivery to the caller app from the messenger service.

```
void cancelClientDelivery()
```

Parameters

None

Return

None

cancelServerNotification()

Cancel server notification.



This API is used internally and should not be called directly by the client app.

```
void cancelServerNotification(java.lang.String packageName,  
                             java.lang.String clientToken)
```

Parameters

- *packageName* indicates the package name of the caller.
- *clientToken* indicates the activated token from the client activation.

Return

None



getClientDeliveryToken()

Return the registered client token for notification delivery.

```
java.lang.String getClientDeliveryToken()
```

Parameters

None

Return

Client token or null if delivery is not activated (that is GCM client ID).

getMessageHandlers()

Register the message handler in the current process.

```
void registerMessageHandler(MessageHandler handler)
```

Parameters

handler indicates an instance of class that implements the MessageHandler interface.

Return

None

getServerNotificationActivated()

Return notification whether or not server notification is activated.

```
boolean getServerNotificationActivated()
```

Parameters

None

Return

true indicates that notification is activated with the server.



isDeliveryTypeSupported()

Find out if the specified delivery type is supported by the underlying implementation.

```
boolean isDeliveryTypeSupported(Messenger.DeliveryType type)
```

Parameters

type indicates the delivery type (refer to *Enum Messenger.DeliveryType* on page 65).

Return

true indicates that the delivery type is supported.

isNotificationActivated()

Find out if server notification is activated to this app.

Call *activateClientDelivery()* to initiate activation if server notification is not activated.

```
boolean isNotificationActivated()
```

Parameters

None

Return

true indicates that delivery is activated.

onMessageReceived()

Invoked by the messenger receiver service when message is received from the underlying message service, such as GCM.

```
void onMessageReceived(java.lang.Object message)
```

Parameters

message indicates a Message object. On Android, it is an instance of Bundle.

Return

None



registerMessageHandler()

Register message handler in the current process.

```
void registerMessageHandler(MessageHandler handler)
```

Parameters

handler indicates an instance of class that implements the MessageHandler interface.

Return

None

removeMessageHandler()

Remove message handler in the current process.

```
void removeMessageHandler(MessageHandler handler)
```

Parameters

handler indicates an instance of class that implements the MessageHandler interface.

Return

None



Enum Messenger.DeliveryType

```
public static enum Messenger.DeliveryType
extends java.lang.Enum<Messenger.DeliveryType>
```

The following table lists and describes delivery type methods and a page reference to each of the method.

Methods	Description	Page
toName		page 65
values()	Return an array containing the constants of this enum type in the order they are declared.	page 65
valueOf()	Return the enum constant of this type with the specified name.	page 66

toName

```
public java.lang.String toName()
```

Parameters

None

Return

None

values

Return an array containing the constants of this enum type in the order they are declared.

```
public static Messenger.DeliveryType[] values()
```

This method may be used to iterate over the constants as follows:

```
for (Messenger.DeliveryType c : Messenger.DeliveryType.values())
    System.out.println(c);
```



Parameters

None

Return

An array containing the constants of this enum type in the order they are declared.

Exception

None

valueOf

Returns the enum constant of this type with the name you specified.



Be sure that the name you entered matches an enum identifier exactly. No whitespace characters are allowed.

```
public static Messenger.DeliveryType valueOf(java.lang.String name)
```

Parameter

- *name* is the name of the enum constant to be returned.

Return

The enum constant with the specified name.

Exception

- `NullPointerException` indicates that the argument is null.
- `IllegalArgumentException` indicates that this enum type has no constant with the specified name.



MagnetActivity

This is an Android wrapper class that extends Activity to support Services startup and data injection.

```
@Dynamic
@ExternalContracts(value=)
public class MagnetActivity
extends Activity
implements ViewContainer
```

By default, implicit data injection takes place only after setContentView(int), setContentView(View) or setContentView(View, android.view.ViewGroup.LayoutParams). However, caller may explicitly do the data injection by calling initializeNow().

If you are using InjectView for injected elements, exercise care when initializeNow() is called. To call initializeNow() before setting the content view, use Provider or Reference based injection. For example:

```
private @InjectView(R.id.someid) javax.inject.Provider<TextView> myTextView;
```

Using this approach, injection will succeed as a deferred operation. Once get() is called on the Provider or Reference, the injection point will be evaluated using findViewById().

The following table lists methods included in MagnetActivity and a page reference to each of the method.

Methods	Page
initializeNow()	page 68
onCreate()	page 68
onSaveInstanceState()	page 68
setContentView(int)	page 68
setContentView(view)	page 68
setContentView(View, android.view.ViewGroup.LayoutParams)	page 68
shouldDeregisterSelfOnDestroy()	page 69
shouldInitializeOnCreate()	page 69



initializeNow()

```
protected void initializeNow()
```

onCreate()

```
protected void onCreate(Bundle savedInstanceState)
```

onSaveInstanceState()

```
protected void onSaveInstanceState(Bundle savedInstanceState)
```

setContentView()

```
public void setContentView(int layoutResID)
```

setContentView()

```
public void setContentView(View v)
```

setContentView()

```
public void setContentView(View view,  
                           android.view.ViewGroup.LayoutParams params)
```



shouldDeregisterSelfOnDestroy()

```
protected boolean shouldDeregisterSelfOnDestroy()
```

shouldInitializeOnCreate()

```
protected boolean shouldInitializeOnCreate()
```

MagnetAndroidService

This is an abstract base for an Android Service.

```
@Dynamic
@ExternalContracts(value=)
public abstract class MagnetAndroidService
extends Service
```

The following table lists methods included in MagnetAndroidService and a page reference to each of the method.

Methods	Page
onCreate()	page 69
onDestroy()	page 70
shouldDeregisterSelfOnDestroy()	page 70

onCreate()

```
public void onCreate()
```



onDestroy()

```
public void onDestroy()
```

shouldDeregisterSelfOnDestroy()

```
protected boolean shouldDeregisterSelfOnDestroy()
```

MagnetApplication

This is an Android wrapper class that provides a central place to start up and shut down the kernel for the Android application.

```
@Dynamic
@ExternalContracts(value=)
public class MagnetApplication
extends Application
```

The following table lists methods included in MagnetAndroidService and a page reference to each of the method.

Methods	Page
onCreate()	page 70
onTerminate()	page 70

onCreate()

```
public void onCreate()
```

onTerminate()

```
public void onTerminate()
```



AndroidSettingsConstants

```
public interface AndroidSettingsConstants
```

Fields	Description	Page
MAGNET_DEV_FACTORY_CONFIGURATION_FILE_NAME	File name for Developer Factory generated configuration properties file.	page 71
MAGNET_SHARED_PREFERENCE_FILE_NAME	Preferences name where Magnet preferences are stored.	page 71
PREFS_FILE_NAME	Internal use only.	page 72
PREFS_KEY_DEFAULT_ENABLE_DATA_ENCRYPTION	Default encryption setting.	page 72
PREFS_KEY_DEFAULT_ENABLE_DEBUG	Internal use only.	page 72
PREFS_KEY_DEFAULT_ENABLE_LOCATION	Default location setting.	page 72
PREFS_KEY_ENABLE_DATA_ENCRYPTION	Enable encryption in local cache when using CallManager.	page 72
PREFS_KEY_ENABLE_DEBUG	Internal use only.	page 73
PREFS_KEY_ENABLE_LOCATION	Enable location service.	page 73
PREFS_KEY_GCM_CLIENT_TOKEN	Internal use only.	page 73
PREFS_KEY_GCM_SENDERID	Internal use only.	page 73
PREFS_KEY_LOCATION_LAST_UPDATE_TIME	Internal use only.	page 73
PREFS_KEY_PUSH_ACCOUNT_MAGNETURI	Internal use only.	page 74
PREFS_KEY_USER_ME_MAGNETURI	Internal use only.	page 74

MAGNET_DEV_FACTORY_CONFIGURATION_FILE_NAME

The file name for the Developer Factory generated configuration properties file.

```
static final String MAGNET_DEV_FACTORY_CONFIGURATION_FILE_NAME
```

MAGNET_SHARED_PREFERENCE_FILE_NAME

Preferences name where Magnet preferences are stored. Use it in call to `Context.getSharedPreferences(String, int)`.



```
static final String MAGNET_SHARED_PREFERENCE_FILE_NAME
```

PREFS_FILE_NAME

Internal use only.

```
static final String PREFS_FILE_NAME
```

PREFS_KEY_DEFAULT_ENABLE_DATA_ENCRYPTION

Default encryption setting.

```
static final boolean PREFS_KEY_DEFAULT_ENABLE_DATA_ENCRYPTION
```

PREFS_KEY_DEFAULT_ENABLE_DEBUG

Internal use only.

```
static final String PREFS_KEY_ENABLE_DEBUG
```

PREFS_KEY_DEFAULT_ENABLE_LOCATION

Default location setting.

```
static final boolean PREFS_KEY_DEFAULT_ENABLE_LOCATION
```

PREFS_KEY_ENABLE_DATA_ENCRYPTION

Enable encryption in local cache when using CallManager.

```
static final String PREFS_KEY_ENABLE_DATA_ENCRYPTION
```



PREFS_KEY_ENABLE_DEBUG

Internal use only.

```
static final String PREFS_KEY_ENABLE_DEBUG
```

PREFS_KEY_ENABLE_LOCATION

Enable location service

```
static final String PREFS_KEY_ENABLE_LOCATION
```

PREFS_KEY_GCM_CLIENT_TOKEN

Internal use only

```
static final String PREFS_KEY_GCM_CLIENT_TOKEN
```

PREFS_KEY_GCM_SENDERID

Internal use only

```
static final String PREFS_KEY_GCM_SENDERID
```

PREFS_KEY_LOCATION_LAST_UPDATE_TIME

Internal use only

```
static final String PREFS_KEY_LOCATION_LAST_UPDATE_TIME
```



PREFS_KEY_PUSH_ACCOUNT_MAGNETURI

Internal use only

```
static final String PREFS_KEY_PUSH_ACCOUNT_MAGNETURI
```

PREFS_KEY_USER_ME_MAGNETURI

Internal use only.

```
static final String PREFS_KEY_USER_ME_MAGNETURI
```



7. Defining Caching and Off-line Mode Services

The controller represents functionality that you write running on your client instance(s). *Class **AsyncCallOptions*** allows you to invoke a request in an off-line mode and caching services remotely from your mobile client.

After you have created your project using the Magnet Developer Factory, an API module is created for your project. The API module represents the JAR of your synchronous and asynchronous Controller interfaces that will be deployed to your client mobile Android app as well as to your server instance(s).

The API JAR is included on the client side directly in the case for Android. Currently, when an Android application starts on the device, and when connectivity information to the server has been established, the mobile app “phones home” obtains the list of controllers and meta information describing those controllers (that is, controllers.json) in order to learn of the available controller interfaces for itself. It then creates remote proxies using these co-located controller interfaces that know how to invoke the remote server. In this way, the controller implementations only exist on the server whereas the controller API interfaces are used solely by the mobile app to call into the server.

Controller

```
public interface CallManager {
    extends com.magnet.controller.spi.Controller
        List<Category> getLocalizedCategories(...);

        void updateCategory(...);
}
```

Requests

A request may be reliable or asynchronous. Constraints are used to control the queue process, and may be applied to both reliable and asynchronous requests.

Reliable Requests

A reliable request is associated with the name of a queue. Since reliable operations are optionally tied to queue names, those are **not** skipped, and queues are processed sequentially. When a constraint associated with the head of the queue is in an allowed state the queue be drained of its elements—meaning that the invocations are being messaged to the server asynchronously and reliably. When a queue name is used but a constraint that is not met will cause the entire key to suspend until network (or other constraint conditions) have changed. Refer to *Making a Reliable Request* on page 93 for code samples.

Async Requests

An asynchronous request indicates that no queue is named explicitly; consequently, a queue is not allowed. When an asynchronous request has a constraint and the constraint is dis-allowed, processing will continue to the next element of the queue. The element is not allowed because the constraint is essentially skipped and will be retried later, provided the request timeout on the entry has not expired the next time around. Refer to *Making an Asynchronous Request* on page 95 for code samples.

Listener

Listener is an instance that runs in the UI thread. After entries are processed and results arrive back from the server, a listener can be optionally notified. Listeners are either transient (*setStateListener()* on page 83) or durable (*setStateListener()* on page 87). A transient listener is used in Async requests whereas a durable listener will survive JVM restarts and is used in Reliable requests.



CallManager

CallManager represents functionality that allows you as the developer to manage the collection of asynchronous calls that have completed or are pending to be submitted to the server. It also allows you to clear cache or force the queues to be rechecked if constraints have changed that warrant reattempting calls from the mobile device to the server.

```
public interface CallManager {
    void clearCache();
    void cancelAllPendingCalls();
    void cancelAllPendingCalls(String queueName);
    Collection<Call<?>> getAllPendingCalls();
    Collection<Call<?>> getAllPendingCalls(String queueName);
    Call<?> getCall(String id);
    void reset();
    void run();
}
```

The following table lists and describes *CallManager* methods and a page reference to each method.

Methods	Description	Page
cancelAllPendingCalls()	Clear all queues of pending requests or the named queue of pending requests when the name of the queue is specified.	page 78
clearCache	Clear all cache results.	page 78
getAllPendingCalls()	Return all pending reliable calls.	page 78
getCall()	Return the a call instance by its unique ID.	page 79
reset()	Use as a shortcut for calling cancelAllPendingCalls() and clearCache().	page 79
run()	Wake up all non-empty thread queues (if asleep) to re-attempt queue processing.	page 80



cancelAllPendingCalls()

Cancel all queues of pending calls. When the name of the queue is specified, only the specified queue is canceled.

```
void cancelAllPendingCalls()
```

```
void cancelAllPendingCalls(String queueName)
```

Parameters

queueName (optional) indicates the name of the queue you want to cancel.

Return

Not applicable.

clearCache()

Clear all cache results.

```
void clearCache()
```

Parameters

None.

Return

Not applicable.

getAllPendingCalls()

Return all pending reliable calls. For a blocking method, you can specify the name of the queue to be returned. If the name of the queue is not specified, a list of all pending calls are returned.

```
List<Call<T>> getAllPendingCalls();
```

```
List<Call<T>> getAllPendingCalls(String queueName);
```

Parameter

- *queueName* (optional) indicates the name of the queue you want to return.



Return

A list of pending calls or a pending call associated with the name of the queue specified.

getCalls()

Return a call instance by its unique ID.



Use this method for reliable calls only.

```
Call<?> getCall(String id);
```

Parameter

- *id* (required) indicates a call UUID associated with the call instance that you want to return.

Return

A call object or null. The return is null if the given ID was invalid, the call has timed out, or the asynchronous call was completed.

reset()

Use as a shortcut for calling the following methods:

- `cancelAllPendingCalls()`
- `clearCache()`

```
void reset()
```

Parameter

None.

Return

Not applicable.



run()

At periodic intervals or when there is a queue submission, all non-empty thread queues are awoken (if asleep) to re-attempt processing. If a queue is blocked by a constraint, and the back-end constraint changes state to allow operational flows that were previously blocked, you can call this method to wake up the Call Manager to re-attempt queue processing before the next schedule.

```
void run()
```

Parameter

None.

Return

Not applicable.

Options

Options is the base interface for asynchronous call and reliable call options, which includes the following abstract classes:

- AsyncCallOptions
- ReliableCallOptions

Class AsyncCallOptions

This class is used for an asynchronous call, which allows the caller to use the cached value, impose a restriction when the call can be invoked, and specify a callback. The caller must use create() to create an instance of the options. If no options are specified, asynchronous (unreliable) call is assumed.

The following example shows the asynchronous call declaration.

```
@Contract
public abstract class AsyncCallOptions
extends Object
implements Options
```

The following table lists and describes AsyncCallOptions methods and a page reference to each method.



Methods	Description	Page
<code>create()</code>	Create an instance of asynchronous call options.	page 81
<code>setCacheAge()</code>	Set an acceptable age and enable caching the result.	page 81
<code>setConstraint()</code>	Invoke the call only when the constraint is met.	page 82
<code>setStateListener()</code>	Set a transient listener for this asynchronous call.	page 83
<code>setToken()</code>	Specify an optional opaque custom token for this call.	page 83

create()

Create an instance of asynchronous call options.

```
public static AsyncCallOptions create()
```

Parameter

None.

Return

An instance of an AsyncCallOptions object.

setCacheAge()

Set an acceptable age and enable caching the result. The call uses the cached value (if available) when its age is within the timeout interval; otherwise, the cached value is discarded and the new result is cached.

If this option is not specified, the cached value is discarded and the new result is not cached. You can specify in the `useCacheIfOffline` parameter whether or not to allow the caller to use the cached value (if available) in an off-line mode despite of the cache age.

```
public abstract AsyncCallOptions setCacheAge(long timeout,
                                             TimeUnit unit,
                                             boolean useCacheIfOffline)
```

Parameters

- *timeout* indicates the timeout value to be set. Setting a value of 0 means to discard the cached value.
- *unit* indicates the time unit value as in HOURS, MINUTES, or SECONDS. (Time units smaller than seconds are not supported.)



- *useCacheIfOffline* indicates whether or not to use the cached value in an off-line mode despite of its age.
 - true indicates that the cached value is used in an off-line mode despite of its age.
 - false indicates otherwise.

Example

```
public abstract AsyncCallOptions setCacheAge("10L",
                                             "TimeUnit.SECONDS",
                                             true)
```

Return

The current instance.

setConstraint()

Specify a constraint that a call must meet before it can be processed and invoke the call only when this constraint condition is met. This will retry the async operation periodically, or when network and/or constraint conditions have changed. It will not, however, survive restarts of the mobile application.

```
public abstract AsyncCallOptions setConstraint(Class<?
extends Constraint> constraint)
```

Parameter

- *constraint* indicates any class from Constraint.

Example

```
AsyncCallOptions options = AsyncCallOptions.create();
options.setStateListener(new StateListener<T> {
    public void onStateChanged(Call<T> call) {
        ...
    }
}).setCacheAge(1, TimeUnit.HOURS,
true).setConstraint(WifiConstraint.class));
mAsyncMyController.getLocalizedCategories(..., options);
```

Return

The current instance.



setStateListener()

Set a transient listener for this asynchronous call. A transient listener does not survive JVM restart. The listener runs in the UI thread.

```
public abstract AsyncCallOptions
setStateListener(StateListener listener)
```

Parameter

- *listener*

Example

```
AsyncCallOptions options = AsyncCallOptions.create();
options.setStateListener(new CallListener<T> {
    public void onStateChanged(Call<T> call) {
        ...
    }
});
mAsyncMyController.getLocalizedCategories(..., options);
```

Return

The current instance.

setToken()

Use a token to identify the call and disallow multiple concurrent calls with the same token.

```
public abstract AsyncCallOptions setToken(String token)
```

Parameter

- *token* indicates a symbol (or description) that the callback listener would use for associations.

Return

The current instance.



Class ReliableCallOptions

A reliable asynchronous call allows the caller to:

- Use the cached value, to queue up a call in a persistent storage even in an off-line mode.
- Execute the calls in sequential manner.
- Impose a restriction when the call can be invoked using constraints.
- Specify a timeout for this call.
- Specify a static callback.

The following example shows the reliable call declaration.

```
@Contract
public abstract class ReliableCallOptions
extends Object
implements Options
```

The following table lists and describes ReliableCallOptions methods and a page reference to each method.

Methods	Description	Page
create()	Create an instance of reliable call options.	page 85
setCacheAge()	Set an acceptable age and enable caching the result.	page 85
setCallTimeout	Set the timeout for this call.	page 86
setConstraint()	Specify a constraint to invoke a queued call	page 86
setQueueName	Specify a name for a queue for a reliable execution.	page 87
setStateListener()	Specify a listener class to be invoked.	page 87
setToken()	Define an opaque token that is use for the callback listener associations.	page 87



create()

Create an instance of reliable call options.

```
public static ReliableCallOptions create()
```

Parameters

None.

Return

An instance of a `ReliableCallOptions` object.

setCacheAge()

Set an acceptable age and enable caching the result. The call uses the cached value (if available) when its age is within the timeout interval; otherwise, the cached value is discarded and the new result is cached.

If this option is not specified, the cached value is discarded and the new result is not cached. You can specify in the `useCacheIfOffline` parameter whether or not to allow the caller to use the cached value (if available) in an off-line mode despite of the cache age.

```
public abstract ReliableCallOptions setCacheAge(long timeout,
                                                TimeUnit unit,
                                                boolean useCacheIfOffline)
```

Parameters

- *timeout* indicates the timeout value to be set. Setting a value of 0 means to discard the cached value.
- *unit* indicates the time unit as in HOURS, MINUTES, or SECONDS. (Time units smaller than seconds are not supported.)
- *useCacheIfOffline* indicates whether or not to use the cached value in an off-line mode despite of its age.
 - true indicates that the cached value is used in an off-line mode despite of its age.
 - false indicates otherwise.

Example

```
public abstract ReliableCallOptions setCacheAge("10L",
                                                "TimeUnit:SECONDS",
                                                true)
```



Return

The current instance.

setCallTimeout

Specify a timeout value when the call object expires between `MyAsyncController.myMethod(...)` and `Call#get()` methods. To survive an off-line mode, specify a longer timeout value.

```
public abstract ReliableCallOptions setCallTimeout(long timeout,  
                                                  TimeUnit unit)
```

Parameters

- *timeout* indicates the timeout value to be set.
- *unit* indicates the time unit as in HOUR, MINUTE, or SECOND.

Example

```
ReliableCallOptions options = ReliableCallOptions.create();  
options.setCallTimeout(10, TimeUnit.MINUTES)  
        .setCacheAge(1, TimeUnit.HOURS, false);  
mAsyncMyController.updateCategory(..., options);
```

Return

The current instance.

setConstraint()

Specify a constraint to invoke a queued call. A queued call can be processed only when the constraint is met. If used in conjunction with a queue name being specified, all subsequent calls to the server will be pending until the constraint gating the head of the queue is allowed, or times out or is canceled.

```
public abstract ReliableCallOptions  
setConstraint(Class<? extends Constraint> constraint)
```

Parameters

- *constraint* indicates any class extends from `Constraint`.

Return

The current instance.



setQueueName

Place a call on a queue for FIFO execution. If this option is not specified, the default queue ({@link #DEFAULT_QUEUE}) will be used. The execution ordering is not guaranteed when the default queue name is applied. If constrained calls must not delay or hinder other calls submitted (constrained or not), then the {@link #DEFAULT_QUEUE} should be used. If concurrent invocation is desired then the calls must be put on different queues.

```
public abstract ReliableCallOptions setQueueName(String queueName)
```

Parameters

- *queueName* indicates the name for this queue.

Return

The current instance.

setStateListener()

Specify a listener class to be invoked when the state of the call changes. An instance of the listener will be created and run in the UI thread. The caller may use the data injection to get the execution context.

```
public abstract ReliableCallOptions  
setStateListener(Class<? extends StateListener> listener)
```

Parameters

- *listener* indicates any class extended from StateListener.

Return

The current instance.

setToken()

Define a token to identify the call and disallow multiple concurrent calls with the same token.

```
public abstract ReliableCallOptions setToken(String token)
```

Parameters

- *token* indicates a symbol (or description) that the callback listener would use for associations.

Return

The current instance.



Constraints

Constraints provide a means to impose conditions that a queue processing a submitted call must meet before it can be processed. Constraints are useful so that processing of the queue can be paused under various network conditions. For instance, some operations can occur only in a WiFi zone.

When Constraint is in an allow state, an element will be processed from the queue. If Constraint is disallowed, the queue is not processed until the Constraint is allowed again, placing the entire queue in a wait state while the disallowed constraint is gating the head of the queue. Once the head of the queue has its constraint lifted (i.e., `isAllowed`), the queue is once again drained of its elements. This process continues for every element in the queue.

The following codes show the interface Constraint declaration.

1

```
@Contract
public interface Constraint
```

Only one method is associated with the Constraint interface: `isAllowed`

IsAllowed

Specify whether or not the constraint is allowed.

```
boolean isAllowed()
```

Return

- True indicates that the constraint is allowed.
- False indicates otherwise.

Example

The following example shows a WiFi constraint is implemented.

```
@Singleton
public class NetworkWifiConstraint implements Constraint {

    @Override
    public boolean isAllowed() {
        return true;
    }
}
```



AsyncController

The Magnet Developer Factory converts the synchronous controller that you write to be async/reliable enabled. Each method returns a `Call<?>` instead of the return value declared in your method.

The following example shows a marker interface for an asynchronous controller.

```
public interface MyAsyncController extends AsyncController {
    Call<T> getLocalizedCategories(..., Options options);
    Call updateCategory(..., Options options);
}
```

Interface Call<T>

This interface represents an asynchronous invocation to a controller. An instance of the call is typically returned by a method call from any `AsyncController` subclass, or `CallManager.getCall(String)`.

`<T>` is the return data type.

```
public interface Call<T>
    extends Future<T>
```

The following table lists and describes `Call<T>` methods and a page reference to each method.

Methods	Description	Page
<code>getId()</code>	Return the global unique ID for this asynchronous call.	page 90
<code>getResultTime()</code>	Return the date and time of the result retrieved from the server.	page 90
<code>getState()</code>	Return the current state of this asynchronous call.	page 90
<code>getToken()</code>	Return the token specified by the caller in the options	page 90
<code>isResultFromCache()</code>	Check whether or not the result is from the cache.	page 91



getId

Return the global unique ID for this asynchronous call. This ID is useful only for the reliable call.

```
String getId()
```

Return

A unique ID.

getResultTime

Return the date and time of the result retrieved from the server.

```
Date getResultExpiryTime()
```

Return

The date and time of the result.

getState

Return the current state of this asynchronous call.

```
Call.State getState()
```

Return

The state of the current call.

getToken

Return the token specified by the caller in the options.

```
String getToken()
```

Return

- A token object when a token is specified.
- Null when no token is specified.



isResultFromCache

Check whether or not the result is from the cache.

```
boolean isResultFromCache()
```

Return

- True indicates that the result is from the cache.
- False indicates otherwise.

Enum Call.State

Provide the enum status.

```
public static enum Call.State
extends Enum<Call.State>
```

The following table lists and describes *Call.State* methods and a page reference to each of the method.

Methods	Description	Page
valueOf()	Returns the enum constant of this type with the specified name.	page 92
values()	Returns an array containing the constants of this enum type in the order they are declared.	page 93



valueOf

Return the enum constant of this type with the specified name.

```
public static Call.State valueOf(String name)
```

Parameter

- *name* is the name of the enum constant to be returned.



Be sure that the name you entered matches an enum identifier exactly. No whitespace characters are allowed.

Return

The enum constant with the specified name.

Throw

- *IllegalArgumentException* when this enum type has no constant with the specified name.
- *NullPointerException* when the argument is null.



values

Return an array containing the constants of this enum type in the order they are declared. Use this method to iterate over the constants as follows:

```
for (Call.State c : Call.State.values())
    System.out.println(c);
```

Return

An array containing the constants of this enum type in the order they are declared. The enum type includes:

- **CANCELLED** indicates that the request has been canceled locally.
- **EXECUTED_HAS_RESULT** indicates that a result is available. Note that the result might in fact be representative of a remote exception.
- **EXECUTING_NO_RESPONSE** indicates that the request has been set, but a response was not received from the server (resulting in an unknown state).
- **FAILED** indicates that the request failed because it cannot be sent for various reasons. Asynchronous calls may fail because of its fail-fast nature.
- **QUEUED** indicates that the reliable request has yet to be processed from the queue.
- **TIMEDOUT** indicates that the reliable request has timed out.

Making a Reliable Request

It is typically the case that invocations from the mobile client occur asynchronously and reliably. They are asynchronous in that the invocation is recorded for later playback to the server on a separate worker thread. They are reliable in that after an invocation has been submitted, it is guaranteed to be run **once**—and only once—on the server side within the request timeout period.

You can make a reliable request with the following specifications:

- cache results and callback (page 94)
- cache results but no callback (page 94)
- No caching or callback (page 94)



Caching Results and Callback

```
...

ReliableCallOptions options = ReliableCallOptions.create();

options.setCallTimeout(10, TimeUnit.MINUTES)
        .setCacheAge(1, TimeUnit.HOURS,
false).setStateListener(MyStateListener.class);

mAsyncCallManager.updateCategory(..., options);

...
```

Caching Results but no Callback

```
...

ReliableCallOptions options = ReliableCallOptions.create();

options.setCallTimeout(10, TimeUnit.MINUTES)
        .setCacheAge(1, TimeUnit.HOURS, false);

mAsyncCallManager.updateCategory(..., options);

...
```

No Caching and No Callback

```
...

ReliableCallOptions options = ReliableCallOptions.create();

options.setRequestTimeout(10, TimeUnit.MINUTES);

mAsyncCallManager.updateCategory(..., options);

...
```



Making an Asynchronous Request

An asynchronous request indicates that no queue is named explicitly; consequently, a queue is not allowed. There might be occasions to make a request without caching the results or when reliability is not a concern. You can make an asynchronous request with the following specifications:

- No caching, reliability, or callback (page 95)
- No caching or reliability but with callback (page 95)
- With caching and callback and no reliability (page 96)

No Caching, Reliability, or Callback

```
...  
mAsyncCallManager.updateCategory(..., null);  
...
```

No Caching or Reliability but With Callback

```
...  
options.setCallListener(new CallListener<T> {  
    public void onStateChanged(Call<T> call) {  
        ...  
    }  
});  
mAsyncCallManager.getLocalizedCategories(..., options);  
...
```



Caching and Callback, but no Reliability

```
...

AsyncCallOptions options = AsyncCallOptions.create();

options.setCallListener(new CallListener<T> {

    public void onStateChanged(Call<T> call) {

        ...

    }

}).setCacheAge(1, TimeUnit.HOURS, true);

mAsyncCallManager.getLocalizedCategories(..., options);

...
```



Applying a Constraint

Constraints are useful so that processing of the queue can be paused under varying network conditions. For instance, some operations should occur only in a WIFI zone. The call fails fast and the response listener is immediately called when the constraint is not met.

The following example shows a constraint is applied to an asynchronous request with caching results and callback.

```
...

AsyncCallOptions options = AsyncCallOptions.create();

options.setStateListener(new StateListener<T> {

    public void onStateChanged(Call<T> call) {

...

    }

}).setCacheAge(1, TimeUnit.HOURS,
true).setConstraint(WifiConstraint.class));

mAsyncCallManager.getLocalizedCategories(..., options);...

...
```



Code Samples

```
private ListActivity myListActivity = ...;

private AsyncCallOptions mAsyncOptions = AsyncCallOptions.create();

private ReliableCallOptions mReliableOptions = ReliabelCallOptions.create();

mAsyncOptions.setCacheAge(1, TimeUnit.DAY, true)

    .setStateListener(new StateListener<List<Category>>(myListActivity) {

        public void onStateChanged(Call<List<Category>> call) {
            if (call.isDone()) {
                List<Category> categories = call.get();
                myListActivity.showList(categories);
            }
        }

    });
Call<List<Category>> call = myAsyncController.getLocalizedCategories(...,
mAsyncOptions);
```



Injecting CallManager to Manage Calls

```

@Inject com.magnet.async.CallManager callManager;

Call reconstitutedCall = callManager.getCall(callId);

callManager.clearPendingCalls();
public class MyStateListener<List<Category>> implements StateListener {

    private MyListActivity mActivity;

    public MyStateListener( MyListActivity activity ) {

        mActivity = activity;
    }
    public void onStateChanged(Call<List<Category>> call) {
        switch (call.getState()) {
            case FAILED:
                // cannot submit the request
                Toast.makeText(...).show();
                break;
            case CANCELLED:
                break;
            case EXECUTED_HAS_RESULT:
                // May throw the app Exception (similar to Future.get())
                List<Category> list = call.get();
                mActivity.showList(list);
                break;
        }
    }
}

```



```
public class MyStateListener<List<Category>> implements StateListener {
    @Inject private Context mContext;
    public void onStateChanged(Call<List<Category> call) {
        switch (call.getState()) {
            case FAILED:
                break;
            case CANCELLED:
                break;
            case QUEUED:
                break;
            case TIMEOUT:
                break;
            case EXECUTED_NO_RESPONSE:
                break;
            case EXECUTED_HAS_RESULT:
                break;
        }
    }
}
```



Using BroadcastReceiver to Change the State

For reliable calls, you can use BroadcastReceiver to change a state by specifying a built-in BroadcastStateListener class to broadcast the state change, as shown in the following example.

```
class MyBroadcastReceiver extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        String callId = intent.getStringExtra(BroadcastStateListener.EXTRA_CALL_ID);
        Call<List<Category>> call = CallManager.getCall(callId);
        switch (call.getState()) {
            case FAILED:
                // send a pending intent
                break;
            case CANCELLED:
                break;
            case TIMEOUT:
                // send a pending intent
                break;
            case QUEUED:
                break;
            case EXECUTED_NO_RESPONSE:
                break;
            case EXECUTED_HAS_RESULT:
                // send a pending intent
                break;
        }
    }
}
```





The Mobile Server and the Mobile App Manager must be compatible to communicate. When Magnet App Server detects an incompatible version running on a mobile device, a notification is sent to the mobile device prompting the user to download and install a newer version of the apk.

The following example shows the implementation. A `@Singleton` contract is declared and a `VersionCheckExceptionHandler` class is created to handle the `VersionCheckException`. A notification appears on user mobile devices and prompts for action.

```
import javax.inject.Singleton;
import com.magnet.opus.rest.client.api.OpusRestExceptionHandler;
import com.magnet.opus.rest.version.checker.api.VersionCheckException;
@Singleton
public class VersionCheckExceptionHandler implements OpusRestExceptionHandler
{
    public synchronized void handleException(Throwable ex) {
        if (ex instanceof VersionCheckException)
        { String msg = ex.getMessage(); /* msg format is something:url */ /* * Tell
        user that the version is incompatible and show the * URL (if available) for the
        new version of apk. */ ... }
        }
    }
}
```



The Magnet App Server currently provides three social controllers that you can include in your app to enable access to third-party services. These controllers include Salesforce, LinkedIn, and Facebook.

Using the Salesforce Controller

The following example shows how to add an activity that calls the SoapController, using @Inject.

```
import com.magnet.sfdc.api.*;
...
@Inject
private Reference<SoapController> myController;
...
try {
    SearchNode search = EntityFactory.createNode(SearchNode.class);
    search.setSearchString("find {4159017000} in All fields returning
contact(id, phone, firstname, lastname, email, title, mobilephone,
mailingstreet, mailingcity, mailingstate, mailingpostalcode), lead(id, phone,
firstname, lastname, email, title, mobilephone, street, city, state,
postalcode), account(id, phone, name)");
    SearchResponseNode response = controllerRef.get().search(search);
    List<SearchRecordNode> contactList =
response.getResult().getSearchRecords();
    AccountNode account = (AccountNode)
response.getResult().getSearchRecords().get(0).getRecord();

    AlertDialog dialog = new
AlertDialog.Builder(UserProfileActivity.this).setTitle("Success").setMessage(
"name: " + account.Name().getElement() + " , phone: " +
account.getPhone().getElement()).show();
} catch (Exception e) {
    AlertDialog dialog = new
AlertDialog.Builder(UserProfileActivity.this).setTitle("Exception").setMessag
e(e.getMessage()).show();
}
```

The following example makes the search() call, which returns contacts, leads, and accounts. The search string specifies the text expression to search for, the scope of fields to search, the list of objects and fields to retrieve.

```
SearchNode search = EntityFactory.createNode(SearchNode.class);

    search.setSearchString("find {" + searchText + "} in All fields
returning contact(id, phone, firstname, lastname, email, title, mobilephone,
mailingstreet, mailingcity, mailingstate, mailingpostalcode), " + "lead(id,
phone, firstname, lastname, email, title, mobilephone, street, city, state,
postalcode), account(id, phone, name)");
SearchResponseNode response = salesForceController.get().search(search);
List<SearchRecordNode> contactList = response.getResult().getSearchRecords();
    for(int i=0; i<contactList.size(); i++) {
        SObjectNode node = contactList.get(i).getRecord();
        if(node instanceof AccountNode) {
            AccountNode account = (AccountNode) node;
            if(account.getName()!=null)
                name = account.getName().getElement();
            if(account.getPhone()!=null)
                phone = account.getPhone().getElement();
        } else if(node instanceof ContactNode) {
            ContactNode contactNode = (ContactNode) node;
            if(contactNode.getFirstName()!=null)
                name = contactNode.getFirstName().getElement() + " " +
contactNode.getLastName().getElement();
            if(contactNode.getEmail()!=null)
                email = contactNode.getEmail().getElement();
            if(contactNode.getPhone()!=null)
                phone = contactNode.getPhone().getElement();
            if(contactNode.getTitle()!=null)
                title = contactNode.getTitle().getElement();
            if(contactNode.getMobilePhone()!=null)
                mobilePhone = contactNode.getMobilePhone().getElement();
            if(contactNode.getMailingStreet()!=null)
                address = contactNode.getMailingStreet().getElement();
            if(contactNode.getMailingCity()!=null)
                address = address + " " +
contactNode.getMailingCity().getElement();
            if(contactNode.getMailingState()!=null)
                address = address + " " +
contactNode.getMailingState().getElement();
            if(contactNode.getMailingPostalCode()!=null)
                address = address + " " +
contactNode.getMailingPostalCode().getElement();
```

