



Kapsel Development

SAP Mobile Platform 3.0

DOCUMENT ID: DC-01-0300-01

LAST REVISED: December 2013

Copyright © 2013 by SAP AG or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries. Please see <http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark> for additional trademark information and notices.

Contents

Kapsel Development	1
Developing Kapsel Applications	3
Setting Up the Development Environment	3
Configuring the Application in the Management Cockpit	7
Defining Applications	7
Defining Back-end Connections for Native and Hybrid Apps	9
Defining Application Authentication	11
Creating an Apache Cordova Project	13
Project Settings	15
Kapsel Plugins	16
Using the Kapsel AppUpdate Plugin	17
Using the Logon Plugin	42
Using the Kapsel AuthProxy Plugin	121
Using the Logger Plugin	178
Using the Kapsel Push Plugin	211
Using the Kapsel EncryptedStorage Plugin	246
Using the Kapsel Settings Plugin	274
Developing a Kapsel Application With OData Online .	288
Creating an OData Application	288
Creating an Application Connection	289
Getting Application Settings	292
Running and Testing Kapsel Applications	294
Client-side Debugging	294
Running the Kapsel Application on Android	295
Running the Kapsel Application on iOS	295
Package and Deploy Kapsel Applications	295
Generating and Uploading Kapsel App Files Using the Command Line Interface	295

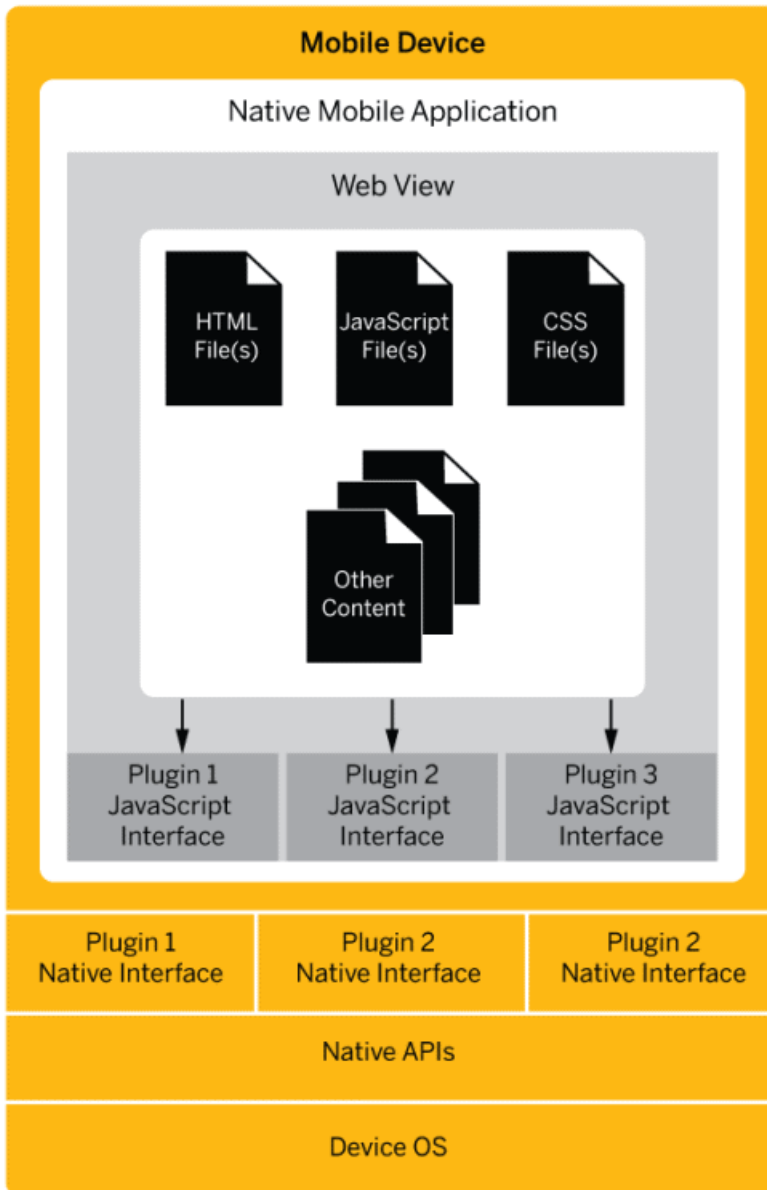
Contents

Preparing the Application for Upload to the Server	296
Uploading and Deploying Hybrid Apps	297
Deploying Hybrid Apps Using the REST API	298
Removing Kapsel Plugins	299
Index	301

Kapsel Development

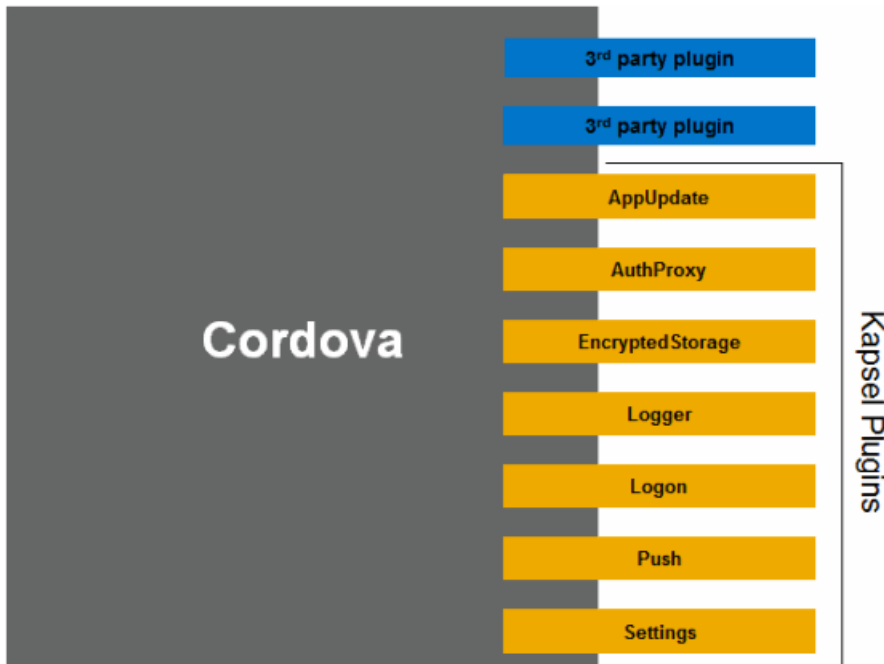
Kapsel is a set of SAP® plugins for Apache Cordova.

Apache Cordova provides a suite of APIs you can use to access native capabilities. The Cordova container provides JavaScript libraries that give you consistent APIs you can call the same way on any supported device. Beginning with Apache Cordova 3.0, the Cordova container is simply a holder in which any APIs and extensions are implemented as plugins. Apache Cordova includes a command line interface for managing Cordova applications and the application development process.



Kapsel leverages the Cordova application container and provides SAP plugins to make the Cordova container enterprise-grade, allowing it to more seamlessly integrate with SAP Mobile Platform Server. The Kapsel plugins provide capabilities like application life cycle management, implementation of a common logon manager and single sign-on (SSO), integration with SAP Mobile Platform Server-based push notifications and so on. Since

Kapsel is implemented without modifying the Cordova container, it is compatible with anything else you develop with Cordova.



Developing Kapsel Applications

Once your application is developed, create a Cordova project and install the Kapsel plugins.

Setting Up the Development Environment

To build Kapsel applications, you must first set up your development environment, which includes installing both SAP Mobile Platform Server, and the SAP Mobile Platform SDK.

Prerequisites

- Verify that you can access SAP Mobile Platform Server from your machine
- If you are using Windows, download and extract Apache Ant and add it to the system variable path, `PATH=%PATH%;C:\apache-ant-<version>\bin`. See <http://ant.apache.org>.

See <http://service.sap.com/pam> to verify that you are using the supported versions for the Kapsel development environment.

Android Requirements

Android tools run on Windows, Linux, and OS X. To build Kapsel apps for Android, you need:

- Java Development Kit (JDK)
- Android SDK

See the Apache Cordova documentation at http://cordova.apache.org/docs/en/3.0.0/guide_platforms_android_index.md.html#Android%20Platform%20Guide for more information about getting started with Android.

Installing the Java SDK

See <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

After installing the Java SDK, define the JAVA_HOME environment variable.

Download the Plugins

Set up the Android Development Environment by downloading the required plugins.

Prerequisites:

- Download the Java Standard Edition Development Kit from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Download the ADT-supported version of Eclipse from <http://www.eclipse.org/downloads/>

1. Start the Eclipse environment.
2. From the **Help** menu, select **Install New Software**.
3. Click **Add**.
4. In the Add Repository dialog, enter a name for the new plugin.
5. Enter one of the following for URL:
 - <https://dl-ssl.google.com/android/eclipse/>
 - <http://dl-ssl.google.com/android/eclipse/>
6. Click **OK**.
7. Select **Developer Tools** and click **Next**.
8. Review the tools to be downloaded.
9. Click **Next**.
10. Read and accept the license agreement and click **Finish**.
11. Once the installation is complete, restart Eclipse.

Installing the ADT Plugin

Follow the instructions for installing the ADT Plugin for Eclipse at <http://developer.android.com/sdk/installing/installing-adt.html>.

If you prefer to work in an IDE other than Eclipse, you do not need to install Eclipse or ADT. You can simply use the Android SDK tools to build and debug your application.

Installing the Google USB Driver

The Google USB Driver for Windows is as an optional SDK component you need only if you are developing on Windows and want to connect a Google Android-powered device (such as a Nexus 7) to your development environment over USB.

Download the Google USB driver package from <http://developer.android.com/sdk/win-usb.html>.

Installing the Android SDK

Install the Android SDK for plugin use with your IDE.

1. Confirm that your system meets the requirements at <http://developer.android.com/sdk/requirements.html>.
2. Download and install the supported version of the Android SDK starter package.
3. Add the Android SDK to your PATH environment variable:
 - On Windows, add <Android SDK Location>\tools to the PATH environment variable
 - On OS X, the command is: `export PATH=$PATH:<path to Android SDK>/tools`
4. Launch the Android SDK Manager and install the Android tools (SDK Tools and SDK Platform-tools) and the Android API.
5. Launch the **Android Virtual Device Manager**, and create an Android virtual device to use as your emulator.

Note: (For offline applications only) Due to limitation on the emulator, you cannot determine the network connection state. For more information on other limitations, see **Emulator Limitations** in <http://developer.android.com/tools/devices/emulator.html#limitations> at the Android Developer Web site.

iOS Requirements

To build Kapsel apps for iOS, you need:

- Mac OS X
- Xcode and Xcode command line tools
- For testing on iOS devices (not the simulator), you need:
 - An Apple Developer account
 - iOS development certificate
 - Provisioning files for each device you are testing with

See the Apache Cordova documentation at http://cordova.apache.org/docs/en/3.0.0/guide_platforms_ios_index.md.html#iOS%20Platform%20Guide for more information about getting started with iOS.

Downloading the Xcode IDE

Download and install Xcode from the Apple Developers Web site.

1. Go to <http://developer.apple.com/downloads/>.

Note: You must be a paying member of the iOS Developer Program. Free members cannot download the supported version.

2. Log in using your Apple Developer credentials.
3. (Optional) To narrow the search scope, unselect all Categories except Developer Tools.
4. Download the appropriate Xcode and SDK combination.

Installing Git

See <http://git-scm.com/book/en/Getting-Started-Installing-Git>.

Note: If you are using a proxy server you must configure git.

```
git config --global http.proxy http://proxy:8080
git config --global https.proxy http://proxy:8080
```

Installing Node.js

Use Node.js v0.10.11 and later, and its package manager, npm, to install Apache Cordova. See <http://nodejs.org/>. You can see the version installed by using the node command: **node -v**.

You must add the Node.js folder to your system PATH.

Note: If you are using a proxy server you must configure npm. At the command prompt, enter:

```
npm config set proxy http://proxy_host:port
npm config set https-proxy http://proxy_host:port
```

Installing the Apache Cordova Command Line Interface

See http://cordova.apache.org/docs/en/3.0.0/guide_cli_index.md.html#The%20Command-line%20Interface. Follow all of the steps in the Cordova command line interface readme.md.

1. Open a command prompt window, and enter:

On Windows: `npm install -g cordova@<latest_supported_version>`

On Mac: `sudo npm install -g cordova@<latest_supported_version>`

For example, to install the Cordova command line interface version 3.0.9, enter:

```
npm install -g cordova@3.0.9
```

-g indicates that Apache Cordova should be installed globally.

Note: If you are installing on Mac and you see a warning message that you are installing globally into a root-only directory, run this command to change the owner of the command line interface installation folder:

```
sudo chown -R user_name /usr/local/lib/node_modules/cordova
```

You can copy the command text from the error message and paste it in at the command prompt at the bottom of the terminal window.

2. On Mac, when prompted, enter your root user password.
3. Verify the Cordova installation by entering this command at the command prompt, or in the terminal window: **cordova -v**

The output shows the Cordova version installed, for example, 3.0.9.

You should also scroll back through the entire installation history shown in the terminal and look for errors to verify the installation was successful.

Installing ios-sim

To allow the Cordova command line to start the iOS simulator on Mac, you must install ios-sim.

1. Download the ios-sim tool files from <https://github.com/phonegap/ios-sim>.
2. Open a terminal window, and enter: `sudo npm install -g ios-sim`
3. When prompted, enter your root user password.
4. Verify the ios-sim installation by entering this command in the terminal window: `ios-sim --version`

The output shows the ios-sim version installed, for example, 1.8.2.

Configuring the Application in the Management Cockpit

Configure the application settings in the Management Cockpit. These settings enable you to monitor and manage your applications.

Prerequisites

- Make sure SAP Mobile Platform Server is installed.
- Make sure the server is started.
- Launch the Management Cockpit.

Task

Defining Applications

Create a new native, hybrid, or Agency application definition. The definition enables you to manage the application using Management Cockpit.

1. In Management Cockpit, select **Applications**, and click **New**.
2. In the New Application window, enter:

Field	Value
ID	<p>Unique identifier for the application in reverse domain notation. This is the application or bundle identifier that is defined or generated during application development. Reverse domain notation is the practice of reversing a registered domain name; for example, the reverse domain notation for an object in <code>sap.com</code> might be <code>MyApp.sap.com</code>). The application identifier:</p> <ul style="list-style-type: none"> • Must be unique. • Must start with an alphabetic character. • Can contain only alphanumeric characters, underscores (<code>_</code>), and periods (<code>.</code>). • Must not include spaces. • Can be up to 64 characters long. <p>Formatting guidelines:</p> <ul style="list-style-type: none"> • SAP recommends that application IDs contain a minimum of two dots (<code>."</code>). For example, this ID is valid: <code>com.sap.mobile.appl</code>. • Application IDs should not start with a dot (<code>."</code>). For example, this ID is invalid: <code>.com.sap.mobile.appl</code>. • Application IDs should not include two consecutive dots (<code>."</code>). For example, this ID is invalid: <code>com..sap.mobile.appl</code>.
Name	<p>Application name. The name:</p> <ul style="list-style-type: none"> • Can contain only alphanumeric characters, spaces, underscores (<code>_</code>), and periods (<code>.</code>). • Can be up to 80 characters long.
Vendor	<p>(Optional) Vendor who developed the application. The vendor name:</p> <ul style="list-style-type: none"> • Can contain only alphanumeric characters, spaces, underscores (<code>_</code>), and periods (<code>.</code>). • Can be up to 255 characters long.
Type	<p>Application type.</p> <ul style="list-style-type: none"> • Native – native iOS and Android applications. • Hybrid – container-based applications, such as Kapsel. • Agentry – metadata-driven application. <hr/> <p>Note: You can configure only one Agentry application per SAP Mobile Platform Server. Once configured, Agentry no longer appears as an option.</p>

Field	Value
Description	(Optional) Short description of the application. The description: <ul style="list-style-type: none"> • Can contain alphanumeric characters. • Can contain most special characters, except for percent signs (%) and ampersands (&). • Can be up to 255 characters long.

3. Click **Save**. Application-related tabs appear, such as Back End, Authentication, Push, and so forth. You are ready to configure the application, based on the application type.

Note: These tabs appear in Management Cockpit only after you define or select an application. The tabs used differ by application type.

Defining Back-end Connections for Native and Hybrid Apps

Define back-end connections for the selected native or hybrid application.

1. From Management Cockpit, select **Applications > Back End**, and enter values for the selected application.

Field	Value
Connection Name <hr/> Note: Appears only when adding a connection under Back-End Connections.	Identifies the back-end connection by name. The connection name: <ul style="list-style-type: none"> • Must be unique. • Must start with an alphabetic character. • Can contain only alphanumeric characters, underscores (_), and periods (.). • Must not include spaces.
Endpoint	The back-end connection URL, or service document URL the application uses to access business data on the back-end system or service. The service document URL is the document destination you assigned to the service in Gateway Management Cockpit. Typical format: <pre>http://host:port/gateway/odata/namespace/Connection_or_ServiceName...</pre> Examples: <pre>http://testapp:65908/help/abc/app1/opg/sdata/TEST-FLIGHT</pre> <pre>http://srvc3333.xyz.com:30003/sap/opu/odata/RMTSAMPLE</pre>

Field	Value
Use System Proxy	(Optional) Whether to use system proxy settings in the SAP Mobile Platform <code>props.ini</code> file to access the back-end system. This setting is typically disabled, because most back-end systems can be accessed on the intranet without a proxy. The setting should only be enabled in unusual cases, where proxy settings are needed to access a remote back-end system outside of the network. When enabled, this particular connection is routed via the settings in <code>props.ini</code> file.
Rewrite URL	(Optional) Whether to mask the back-end URL with the equivalent SAP Mobile Platform Server URL. This is necessary to ensure the client makes all requests via SAP Mobile Platform Server and directly to the back end. Rewriting the URL also ensures that client applications need not do any additional steps to make requests to the back end via SAP Mobile Platform Server. If enabled, the back-end URL is rewritten with the SAP Mobile Platform Server URL. By default, the property is enabled.
Allow anonymous connections	<p>(Optional) Whether to enable anonymous access. This means the application user can access the application without entering a user name and password. However, the back-end system still requires log on credentials to access the data, whether it is a read-only user, or a back-end user with specific roles. If enabled, enter the log on credential values used to access the back-end system:</p> <ul style="list-style-type: none"> • User name – supply the user name for the back-end system. • Password – supply the password for the back-end system. <p>If disabled, you do not need to provide these credentials. By default, the property is disabled.</p> <hr/> <p>Note: If you use Allow Anonymous Connections for a native OData application, do not assign the No Authentication Challenge security profile to the application, or the anonymous OData requests will not be sent (Status code: 401 is reported).</p>
Maximum Connections	<p>The number of back-end connections that are available for connection pooling for this application. The larger the pool, the larger the number of possible parallel connections to this specific connection. The default is 500 connections. Factors to consider when resetting this property:</p> <ul style="list-style-type: none"> • The expected number of concurrent users of the application. • The load acceptable to the back-end system. • The load that the underlying hardware and network can handle. <hr/> <p>Note: The maximum connections can be increased only if SAP Mobile Platform Server hardware can support the additional parallel connections, and if the underlying hardware and network infrastructure can handle it.</p>

Field	Value
Certificate Alias	The name under which the administrator has imported the certificate key-pair in the <code>smp_keystore</code> file. The alias must be set when the back-end URL is <code>https://</code> , and the back-end server requires mutual authentication. There are conditions when <code>https</code> is used but the server does not require a client certificate. This certificate alias is required when the back end requires mutual SSL connectivity. Use the alias of a certificate stored in the SAP Mobile Platform Server keystore. SAP recommends that the CN value of the generated certificate be the fully qualified domain name of SAP Mobile Platform Server.

2. (Optional) Under Back-end Connections, view additional connections, or add new connections.
 - a) Click **New**, to add additional back-end connections in the server.
 - b) Enter values for the new back-end connection, using the values shown above.
 - c) Click **Save**. The new connection is added to the list.

Administrator maintains a list of server-level back-end connections (it includes all the connections in the SAP Mobile Platform Server) and application specific back-end connections. Application specific back-end connections are the connections enabled for an application. Users registered to an application can access only these back-end connections. Request-response to a back end connection that is not enabled for an application is not allowed (throws 403, "Forbidden" error).

By default, these additional back-end connections are enabled for an application.

Back-end connection is displayed in the list.

3. Select the **Application-specific Connections** from the drop-down to show the back-end connections that are enabled for a particular application.

You can view the **Server-level Connections** and enable the connections for this application using the checkbox.

Note: Multiple back ends can be authenticated using various options of authenticating requests available in security configuration.

Defining Application Authentication

Assign a security profile to the selected application. The security profile defines parameters for how the server authenticates the user during onboarding, and request-response interactions.

Prerequisites

Note: You must configure security profiles for application authentication before you can complete this step.

Task

Security profiles are comprised of one or more authentication providers. These authentication providers can be shared across multiple security profiles, and can be modified in Management Cockpit. You can stack multiple providers to take advantage of features in the order you chose; the Control Flag must be set for each enabled security provider in the stack.

1. From Management Cockpit, select **Applications > Authentication**.
2. Click **Existing Profile**.

Note: You can also create a new profile.

3. In Name, select a security profile name from the list. The name appears under Security Profile Properties, and one or more security providers appear under Authentication Providers.
4. Under Security Profile Properties, enter values.

Field	Value
Name	A unique name for the application authentication profile.
Check Impersonation	(Optional) In token-based authentication, whether to allow authentication to succeed when the user name presented cannot be matched against any of the user names validated in the login modules. To prevent the user authentication from succeeding in this scenario, the property is enabled by default.

5. Under Authentication Providers, you can select a security profile URL to view its settings. To change its settings, you must modify it through **Settings > Security Profiles**.

Kapsel Security Matrix

Use one of the supported security configurations to secure your applications.

Security Configuration	Implemented Using	Security Provider
Basic authentication with HTTP	Kapsel Logon plugin	No Authentication Challenge
Basic authentication with HTTPS	Kapsel Logon plugin	No Authentication Challenge
Mutual authentication with HTTPS using a certificate	Kapsel Logon plugin, Client Hub, Afaria	X.509 User Certificate
SiteMinder (non-network edge)	Kapsel Logon plugin	HTTP/HTTPS Authentication
SiteMinder network edge (reverse proxy)	Kapsel Logon plugin	Populate JAAS Subject From ClientHTTP/HTTPS Authentication

Security Configuration	Implemented Using	Security Provider
SSO2 token (HTTP and HTTPS)	Kapsel Logon plugin, Kapsel AuthProxy plugin	HTTP/HTTPS Authentication
SSO passcode with Client Hub	Kapsel Logon plugin, Client Hub	System Login (Admin Only)
User name and password using Client Hub	Kapsel Logon plugin, Client Hub	System Login (Admin Only)
Basic authentication with LDAP back end	Kapsel Logon plugin	Directory Service (LDAP/AD)
Encrypted storage	Kapsel EncryptedStorage plugin	Any
Data Vault	Kapsel Logon plugin	Any

Creating an Apache Cordova Project

To create projects for use with Kapsel, use the Cordova command line tool.

Prerequisites

Set up your development environment.

Task

You must run the commands from a Windows command prompt, or a terminal window on iOS. See http://cordova.apache.org/docs/en/3.0.0/guide_cli_index.md.html#The%20Command-line%20Interface.

1. Create a folder to hold your Kapsel Cordova projects.

For example, on Windows, C:\Documents and Settings\

2. Open a Windows command prompt or terminal and navigate into the project folder you created.

3. At the command prompt, enter:

On Windows: `cordova -d create <Project_Folder>
<Application_ID> <Application Name>`

On Mac: `cordova -d create ~<Project_Folder> <Application_ID>
<Application Name>`

The -d flag indicates debug output and is optional.

This may take a few minutes to complete, as an initial download of the template project that is used is downloaded to C:\Users\user\.cordova on Windows, or ~/users/user/.cordova on Mac.

The parameters are:

- (Required) *<Project_Folder>* – the directory to generate for the project.
- (Optional) *<Application_ID>* – must match the Application ID as configured on SAP Mobile Platform Server for the application, which is reverse-domain style, for example, com.sap.kapsel.

Note: *<Application_ID>* cannot be too simple. For example, you can have "a.b" for an ID, but you cannot have "MyApplicationId." The ID is used as the package name (name space) for the application and it must be at least two pieces separated by a period, otherwise, you will get build errors.

- (Optional) *<Application_Name>* – name for the application.

In this example, you create a project folder named LogonDemo in the Kapsel_Projects directory. The Application ID is "com.mycompany.logon" and the application name is "LogonDemo." Running **cordova -d** allows you to see the progress of the project creation.

```
cordova -d create ~\Kapsel_Projects\LogonDemo
com.mycompany.logon LogonDemo
```

Your new project includes scripts to build, emulate, and deploy your application.

Note: All of the Cordova command line interface commands operate against the current folder. The **create** command creates a folder structure for your Cordova projects while the remaining commands must be issued from within the project folder created by create.

4. To add the platform, change to the folder you created in the previous step:

```
cd <~Project_Name>
```

This OS X example adds the Android and iOS platforms, creating both an Xcode project and an Android project.

```
cd ~\Kapsel_Projects\LogonDemo
cordova platform add ios android
```

Note: Android is supported on both Windows and OS X, but iOS is supported only on OS X.

Note: You must add the platform before you add any Kapsel plugins.

The project directory structure is similar to this:

```
LogonDemo/
|-- .cordova/
|-- merges/
| |-- android/
| `-- ios/
|-- platforms/
| |-- android/
```

```
| `-- ios/
|-- plugins/
|-- www/
`-- config.xml
```

- `.cordova` – identifies the project as a Cordova project. The command line interface uses this folder for storing its lazy loaded files. The folder is located immediately under your user’s home folder (On Windows, `c:\users\user_name\`, and on Macintosh, `/users/user_name/.cordova`).
 - `merges` – contains your Web application assets, such as HTML, CSS, and JavaScript files within platform-specific subfolders. Files in this folder override matching files in the `www/` folder for each respective platform.
 - `www` – this folder contains the main HTML, CSS, and JavaScript assets for your application. The `config.xml` file contains meta data and native application information needed to generate the application. The `index.html` file is the default page of the application. Once you finish editing your project's files, update the platform specific files using the **cordova -d -prepare** command.
 - `platforms` – native application project structures are contained in subfolders for the platforms you added to your application.
5. (Optional) You can test your Cordova project by opening it in the respective development environment, for example, Xcode or Eclipse with the ADT plugins, and running it on the simulator or emulator.
 6. Add the plugins. For example, to add the Cordova console plugin and the Kapsel Logon plugin on Windows, enter:

```
cordova plugin add https://git-wip-us.apache.org/repos/asf/
cordova-plugin-console.git
cordova -d plugin add C:\SAP\MobileSDK3\KapselSDK\plugins\logon
```

Note: The path you enter to the Kapsel plugin must be the absolute path (not relative path).

7. Edit the Web application content in the project's `www` folder and use the **cordova prepare** command to copy that content into the Android and iOS project folders:

```
cordova -d prepare android
cordova -d prepare ios
```

Project Settings

To set application configuration parameters, use the Cordova platform-independent `config.xml` file.

To modify application metadata, edit the `config.xml` file. The `config.xml` file is located in the `www` directory in your project. For information about the project settings for each platform, see http://cordova.apache.org/docs/en/3.0.0/config_ref_index.md.html#Configuration%20Reference.

Kapsel Plugins

Developers use one or more Kapsel plugins in Cordova applications to add SAP Mobile Platform awareness and capabilities to the application. The plugins that you use vary depending on your application's requirements. As they are standard Cordova plugins, manage Kapsel plugins in a Cordova project using the standard Cordova CLI plugin commands.

Kapsel Plugin	Use
AppUpdate	<p>(Required) As the Kapsel lifecycle management plugin, AppUpdate manages application update downloads and installs updates to the Kapsel application. The AppUpdate plugin initiates the check for an update when the application starts, and when it resumes after being suspended. You can also start an app update manually, if required.</p> <p>AppUpdate requires the Logon plugin; the two plugins are installed together.</p>
Logon	<p>Manages user onboarding and the authentication process for SAP Mobile Platform applications. Most other Kapsel plugins use capabilities that this plugin exposes. The plugin interfaces with the SAP Afaria® client as well as the Client Hub application to help manage authentication and single sign-on.</p> <p>You can install this plugin standalone, or it is automatically installed with AppUpdate.</p>
AuthProxy	<p>Provides capabilities that are used in certain security scenarios such as mutual authentication and in SiteMinder environments.</p>
Logger	<p>Lets you have an application write entries to a local log, which can be uploaded to the SAP Mobile Platform Server for analysis. The SAP Mobile Platform administrator can manage setting the application log remotely from the server and upload device logs to the server without user intervention.</p>
Push	<p>Manages the process of registering for push requests as well as exposes events that help you code an application to respond to push notifications. Once the push registration is completed, the plugin uses the Settings plugin to exchange application settings information with SAP Mobile Platform Server so it knows how to manage delivery of push notifications to the application.</p>

Kapsel Plugin	Use
EncryptedStorage	Adds an encrypted persistent store (key/value pair) to a Cordova application, which allows you to build an application that securely stores application data while offline, or while the application is not running. Unlike the built-in local storage, EncryptedStorage is nonblocking.
Settings	Required if you are using the Push plugin. Manages the exchange of settings information between the Kapsel app and the SAP Mobile Platform server. Used by the Push plugin.

Using the Kapsel AppUpdate Plugin

The AppUpdate plugin provides server-based updates to the Web application content that is running in the Kapsel application.

AppUpdate Plugin Overview

The AppUpdate plugin lets an administrator remotely update the contents in the `www` folder of a deployed Kapsel application.

This means that updates to the Web application content only, which does not include application bundle contents outside the `www` folder, do not require corresponding updates to the native application bundle on the end-users' devices.

Note: When you update Web content for applications that are distributed through a public app store, you must adhere to the policies of the app store provider, even though you do not need to go through the formal review process. Do not include updates to content that violates the terms of the app store content review policies, or change the functionality of the application.

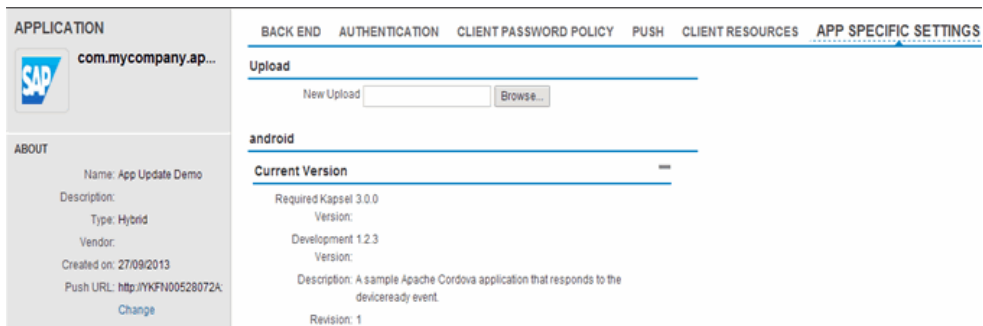
The AppUpdate plugin requires no developer programming, but includes a JavaScript API for customizing the way that application updates occur. The AppUpdate plugin operates in a default mode unless you handle the provided callback APIs.

Configuration Parameters

These configuration parameters are mapped between the Management Cockpit and the `www` folder's `config.xml` file. See *Managing Update Versions and Revisions* for information about usage.

Management Cockpit	config.xml File	Example Value
Revision	hybridapprevision	1

This shows an example of app-specific settings configuration for a sample app in Management Cockpit.



The settings in Management Cockpit are mapped to: `Sample <AppDirectory>/www/config.xml` configuration `<preference name="hybridapprevision" value="1" />`

Note: The revision and development versions on SAP Mobile Platform Server are independent values. The Development Version is an optional value for the administrators' convenience, and is not used by the AppUpdate plugin. Revisions are auto incremented upon each update of the `www` folder archive on the server, regardless of whether the development version changes.

Update Flow

1. The administrator uploads a new archive of the `www` folder contents to SAP Mobile Platform Server, where he or she can update one or more platform versions of the `www` folder in an operation. The administrator specifies the minimum version of Kapsel required for the update, and the development version (for example, the build version). The SAP Mobile Platform Server auto increments the revision number when the administrator clicks **Deploy** or **Deploy All**.
For details about these administrator tasks, as well as information on the underlying REST API that you can use to automate update uploads, see *administrator Guide > Application Administration > Deploying Applications > Defining Application-Specific Settings > Uploading and Deploying Hybrid Apps*.
2. The Kapsel application with the AppUpdate plugin checks with SAP Mobile Platform Server to see if there is a later revision of the `www` folder contents available. If the server has a revision that is greater than the currently downloaded revision, the updated `www` folder is downloaded. SAP Mobile Platform Server and the AppUpdate plugin support delta downloads between revision numbers for a development version of the `www` folder archive. See *Managing Update Versions and Revisions*.
3. If an update to the native Kapsel application bundle is distributed, the currently downloaded revisions of the `www` folder contents are retained through the update. When a newer revision is available on SAP Mobile Platform Server, the delta of the `www` folder contents between the on-device and server revision numbers are downloaded to the Kapsel application. For application bundle updates with very large changes to the `www` folder contents, you can specify a **hybridapprevision** parameter in the application bundle's `config.xml` matching that revision on SAP Mobile Platform Server, so that a delta

download takes place. The `www` folder contents in the Kapsel application bundle are then read, as if from a downloaded revision. Future revisions to the `www` folder contents uploaded to the SAP Mobile Platform Server are downloaded normally by the AppUpdate plugin. See *Managing Update Versions and Revisions*.

4. Once an update is downloaded by the AppUpdate plugin, there are a series of configurable behaviors for handling the end-user experience, and for when the update is applied. The default behavior is to display a modal alert to the user with options to accept or defer updates. If the end user accepts the update, the Web application session is restarted within the Kapsel application container, and the new version is loaded.

Example 1: User Accepts App Update

1. The `AppUpdate` function starts and triggers any required log on process.
2. Checking event is fired by `AppUpdate`.
3. `AppUpdate` finds that an update is available on the server, and the downloading event fires.
4. Updates finish downloading.
5. The `sap.AppUpdate.onupdateready` function is triggered.
6. A prompt asks the user to reload the application.
7. The user accepts the prompt.
8. The `sap.AppUpdate.reloadApp` function is called and the updated application loads.

Example 2: User Defers Update Action

1. The `AppUpdate` function starts and triggers any required log on process.
2. Checking event is fired by `AppUpdate`.
3. `AppUpdate` finds that an update is available on the server, and the downloading event is fired.
4. Updates finish downloading.
5. The `sap.AppUpdate.onupdateready` function is triggered.
6. A prompt asks the user to reload the application.
7. The user cancels the prompt.
8. The `sap.AppUpdate.onupdateready` function is triggered the next time the application is resumed or started.

Configuring the AppUpdate User Experience

You can modify the user experience of the update event by using the `onUpdateReady()` function in the JavaScript application code. These modifications include managing the UI that is shown to the user, text strings, look and feel, position of alert, and so on. You can also add behaviors such as storing a timestamp of the last time the end user was prompted for an update, then waiting for some fixed period of time, such as a week, before again prompting the user to update.

Note: Ensure that any code written for the `onUpdateReady()` function that defers, or otherwise overrides, default update life cycle includes an appropriate recovery method, and does not permanently turn off updates.

Example of Overriding Default Update Behavior

You can assign a custom function to the `onUpdateReady()` event to override default update behavior and force an update that does not ask the user to confirm it. It can either go immediately, or the Administrator can set a date by which it goes.

To do this, add a custom function to `onUpdateReady()`, for example:

```
sap.AppUpdate.onupdateready = myCustomAppUpdateFunction
```

Then, in that custom function, control the update process in whatever way you want. For example, to automatically load the update without first prompting the user for permission, you can add something similar to this:

```
function myCustomAppUpdateFunction = {
  // No notification just reload
  console.log("Applying application update...");
  sap.AppUpdate.reloadApp();
}
```

To use your own custom prompt to warn the user that the app is ready to update, you can do something similar to this:

```
function myCustomAppUpdateFunction = (e){
  console.log("Confirming application update...");
  navigator.notification.confirm('Do you want to install the latest
  application update?', doAppUpdateContinue, 'Please confirm', 'Yes,
  No');
}

function doAppUpdateContinue(buttonNum) {
  if (buttonNum==1) {
    console.log("Applying application update...");
    sap.AppUpdate.reloadApp();
  }
};
```

Managing Update Versions and Revisions

SAP Mobile Platform Server with the AppUpdate plugin supports both full updates (a complete download of the `www` folder archive contents on the server) and delta updates (only changed files are downloaded to the device).

These rules govern how updates are downloaded to the device:

1. If the **hybridappreversion** parameter in `config.xml` = 0, or is omitted, the AppUpdate plugin downloads the complete `www` folder archive from the server the first time the device connects. There is no delta comparison between the server revision and the initial copy on the device—the full `www` folder is downloaded, and becomes **hybridappreversion**=`<current_server_revision_number>` on the device.

The initial copy from the application bundle functions normally, until the time that AppUpdate downloads the first revision from the server.

In other words, since the server's auto incremented Revision value starts at 1, a **hybridapprevision** value of 0, or an empty value in the `config.xml` tells the AppUpdate plugin that it is working with the application bundle copy.

2. If the **hybridapprevision** on the device (either set in `config.xml`, or managed by AppUpdate plugin) is greater than 0, and there is a newer revision on the server, then the AppUpdate plugin downloads only changed, new, or deleted resources—a delta update. The delta calculations are executed by SAP Mobile Platform Server before a request from the AppUpdate plugin, and are maintained for updating from any available historical revision on the server to the current revision.

This table shows an example of the update behavior. A valid update path is any distance to the right on the matrix.

Device hybridapprevision	Server Revision					
	1.2.3/1	1.2.3/2	1.2.3/3	1.3.0/4	1.5.1/5	2.0.0/6
0	Full	Full	Full	Full	Full	Full
1		Delta	Delta	Delta	Delta	Delta
2			Delta	Delta	Delta	Delta
3				Delta	Delta	Delta
4					Delta	Delta
5						Delta

Best Practices

- For most smaller Web applications, you should simply omit the **hybridapprevision** parameter from the `config.xml`. This ensures that the revision numbering on-device and on the server is correctly aligned. The only 'full' download occurs upon the Kapsel application bundle's installation and initialization—all subsequent downloads will be deltas.
- For large Web applications (tens of MBs or greater), setting the **hybridapprevision** parameter in the `config.xml` can greatly reduce the download volume. You should ensure that the value on-device matches the correct value for the server. Since the values on the server are auto incremented, it may be advisable when setting this parameter to complete the upload on the server before packaging and distributing the Kapsel application bundle. This ensures that the correct value is used.

Adding the AppUpdate Plugin

To install the AppUpdate plugin, use the Cordova command line interface.

Prerequisites

- Set up the development environment.
- Create your Cordova Project.
- Add your OS platforms.

Task

Note: The AppUpdate plugin has dependencies on the Logon plugin, as well as some Cordova plugins. These are automatically added to your project when you add the AppUpdate plugin.

1. Add the AppUpdate plugin by entering the following at the command prompt, or terminal:

On Windows:

```
cordova -d plugin add <SDK_HOME>\MobileSDK3\KapselSDK  
\plugins\appupdate
```

On Mac:

```
cordova -d plugin add ~<SDK_HOME>/MobileSDK3/KapselSDK/  
plugins/appupdate
```

Note: The path you enter to the Kapsel plugin must be the absolute path (not relative path).

2. (Optional) To see a list of installed plugins in your Cordova project, open a command prompt or terminal window, navigate to your Cordova project folder, and enter:

```
cordova plugins
```

The Cordova command line interface returns a JSON array showing installed plugins, for example:

```
[ 'org.apache.cordova.core.camera',  
'org.apache.cordova.core.device-motion',  
'org.apache.cordova.core.file' ]
```

In this example, the Cordova project has the Cordova core Camera, Accelerator (device-motion), and File plugins installed.

3. Modify the files in the www folder for the project as necessary, then copy them to the platform directories by running:

```
cordova -d prepare android  
cordova -d prepare ios
```

4. Use the Android IDE or Xcode to deploy and run the project.

Note: If you are using an iOS device, remember to add the "clienthubEntitlements" to the Keychain Groups in the Entitlement section in Xcode.

Kapsel AppUpdate API Reference

The Kapsel AppUpdate API Reference provides usage information for AppUpdate API classes and methods, as well as provides sample source code.

AppUpdate namespace

Used to provide server-based updates to the application content.

The AppUpdate plugin updates the contents of the www folder of deployed Kapsel applications. After an application successfully does a logon to an SAP Mobile Platform 3 server, the AppUpdate plugin is able to download an available update. See Uploading Hybrid Apps in user documentation for information on how to upload an update to SAP Mobile Platform 3 server.

After an update is completely downloaded, the application user is prompted to install the update and restart the application. They can decline if they wish.

Once an update is installed, the application's revision number is updated.

Adding and Removing the AppUpdate Plugin

The AppUpdate plugin is added and removed using the *Cordova CLI*.

To add the AppUpdate plugin to your project, use the following command:

```
cordova plugin add <path to directory containing Kapsel plugins>\appupdate
```

To remove the AppUpdate plugin from your project, use the following command:

```
cordova plugin rm com.sap.mp.cordova.plugins.appupdate
```

Hybrid App Revision Preference

This is an optional preference that tells the AppUpdate plugin if the local assets are uploaded to the server, and at what number. If this preference is not provided, the default revision is 0. In your config.xml file you can add the following preference:

```
<preference name="hybridapprevision" value="1" />
```

This means that the local assets in your www folder are uploaded to the server and the server is reporting revision 1 for them. This allows the application to receive a delta update when revision 2 is available instead of a full update.

Caveats

It is important to test that your update has valid HTML, Javascript, and CSS. Otherwise, the update could prevent the application from functioning correctly, and may no longer be updateable. You can test the updated application in a separate simulator or additional test device. You can also validate your Javascript with tools like *JSLint*, or *JSHint*. You can validate CSS with *CSS Lint*.

Methods

Name	Description
<i>addEventListener(eventname, f)</i> on page 25	Add a listener for an AppUpdate event.
<i>reloadApp()</i> on page 26	Replaces the app resources with any newly downloaded resources.
<i>removeEventListener(eventname, f)</i> on page 26	Removes a listener for an AppUpdate event.

<i>reset()</i> on page 27	Removes all local updates and loads the original web assets bundled with the app.
<i>update()</i> on page 27	Force an update check.

Events

Name	Description
<i>checking</i> on page 27	Event fired when AppUpdate is checking for an update.
<i>downloading</i> on page 28	Event fired when AppUpdate has found an update and is starting the download.
<i>error</i> on page 28	Event fired when AppUpdate encounters an error while checking for an update or downloading an update. The status code and status message are provided with this event.
<i>noupdate</i> on page 29	Event fired when AppUpdate finds no available updates on server.
<i>updateready</i> on page 29	Event fired when AppUpdate has a newly downloaded update available.

Source

appupdate.js, line 84 on page 34.

addEventListener(eventname, f) method

Add a listener for an AppUpdate event.

See events for available event names.

Syntax

```
<static> addEventListener( eventname, f )
```

Parameters

Name	Type	Description
<i>eventname</i>	string	Name of the app update event.
<i>f</i>	function	Function to call when event is fired.

Example

```
sap.AppUpdate.addEventListener('checking', function(e) {  
    console.log("Checking for update");  
});
```

Source

appupdate.js, line 133 on page 35.

reloadApp() method

Replaces the app resources with any newly downloaded resources.

Syntax

<static> reloadApp()

Example

```
sap.AppUpdate.reloadApp();
```

Source

appupdate.js, line 108 on page 35.

removeEventListener(eventname, f) method

Removes a listener for an AppUpdate event.

See events for available event names.

Syntax

<static> removeEventListener(*eventname*, *f*)

Parameters

Name	Type	Description
<i>eventname</i>	string	Name of the app update event.
<i>f</i>	function	Function that was registered.

Example

```
// Adding the listener  
var listener = function(e) {  
    console.log("Checking for update");  
};  
sap.AppUpdate.addEventListener('checking', listener);  
  
// Removing the listener  
sap.AppUpdate.removeEventListener('checking', listener);
```

Source

appupdate.js, line 153 on page 36.

reset() method

Removes all local updates and loads the original web assets bundled with the app.

Call this after delete registration. Reset calls error callback if called during the update process.

Syntax

```
<static> reset()
```

Example

```
sap.Logon.core.deleteRegistration(function() {
    sap.AppUpdate.reset();
}, function() {});
```

Source

appupdate.js, line 120 on page 35.

update() method

Force an update check.

By default updates are done automatically during logon and resume. See events for what will be fired during this process.

Syntax

```
<static> update()
```

Example

```
sap.AppUpdate.update();
```

Source

appupdate.js, line 91 on page 34.

checking event

Event fired when AppUpdate is checking for an update.

Properties

Name	Type	Default	Description
<i>type</i>	string	undefined	The name of the event. Value will be checking.

Type
object

Example

```
sap.AppUpdate.addEventListener('checking', function(e) {  
    console.log("Checking for update");  
});
```

Source

appupdate.js, line 159 on page 36.

downloading event

Event fired when AppUpdate has found an update and is starting the download.

Properties

Name	Type	Default	Description
<i>type</i>	string	undefined	The name of the event. Value will be downloading.

Type
object

Example

```
sap.AppUpdate.addEventListener('downloading', function(e) {  
    console.log("Downloading update");  
});
```

Source

appupdate.js, line 163 on page 37.

error event

Event fired when AppUpdate encounters an error while checking for an update or downloading an update. The status code and status message are provided with this event.

Properties

Name	Type	Default	Description
<i>type</i>	string	undefined	The name of the event. Value will be error.

<i>statusCode</i>	int	undefined	The http error code.
<i>statusMessage</i>	string	undefined	The http status message.

Type
object

Example

```
sap.AppUpdate.addEventListener('error', function(e) {
    console.log("Error downloading update. statusCode: " +
e.statusCode + " statusMessage: " + e.statusMessage);
});
```

Source

appupdate.js, line 165 on page 37.

noupdate event

Event fired when AppUpdate finds no available updates on server.

Properties

Name	Type	Default	Description
<i>type</i>	string	undefined	The name of the event. Value will be <i>noupdate</i> .

Type
object

Example

```
sap.AppUpdate.addEventListener('noupdate', function(e) {
    console.log("No update");
});
```

Source

appupdate.js, line 161 on page 36.

updateready event

Event fired when AppUpdate has a newly downloaded update available.

A default handler is already added to `sap.AppUpdate.onupdateready` that will ask the user to reload the app. When handling this event you should call `sap.AppUpdate.reloadApp()` to apply the downloaded update.

Properties

Name	Type	Default	Description
<i>type</i>	string	undefined	The name of the event. Value will be updateready.
<i>revision</i>	int	undefined	The revision that was downloaded.

Type
object

Example

```
// This will listen for updateready event.
// Note: Use sap.AppUpdate.onupdateready if you want to override the
// default handler.
sap.AppUpdate.addEventListener('updateready', function(e) {
    console.log("Update ready");
});

// Override default handler so that we automatically load the update
// without first prompting the user for permission,
sap.AppUpdate.onupdateready = function(e) {
    // No notification just reload
    console.log("Apply application update...");
    sap.AppUpdate.reloadApp();
};

// Override default handler with custom prompt to warn the user that
// the application is ready to update.
sap.AppUpdate.onupdateready = function() {
    console.log("Confirming application update...");
    navigator.notification.confirm('Update Available',
        function(buttonIndex) {
            if (buttonIndex === 2) {
                console.log("Applying application update...");
                sap.AppUpdate.reloadApp();
            }
        },
        "Update", ["Later", "Relaunch Now"]);
};
```

Source

appupdate.js, line 167 on page 37.

Source code

appupdate.js

```
1 // ${project.version}
2 var exec = require('cordova/exec'),
3     channel = require('cordova/channel'),
4     logonFired = false, // Flag to determine if logon manager
    is done
5     promptActive = false, // Flag to prevent prompt from
    displaying more than once
6     bundle = null; // Internationalization. Loaded with device
    ready
7
8
9 // Event channels for AppUpdate
10 var channels = {
11     'checking': channel.create('checking'),
12     'noupdate': channel.create('noupdate'),
13     'downloading': channel.create('downloading'),
14     'error': channel.create('error'),
15     'updateready': channel.create('updateready')
16 };
17
18 // Holds the dom 0 handlers that are registered for the
    channels
19 var domZeroHandlers = {};
20
21 // Private callback that plugin calls for events
22 var _eventHandler = function (event) {
23     if (event.type) {
24         if (event.type in channels) {
25             channels[event.type].fire(event);
26         }
27     }
28 };
```

Kapsel Development

```
29
30     /** @namespace sap */
31
32     /**
33      * Used to provide server-based updates to the application
34      * content.
35      * <br/><br/>
36      * The AppUpdate plugin updates the contents of the www folder
37      * of deployed Kapsel
38      * applications. After an application successfully does a
39      * logon to an SAP Mobile Platform 3
40      * server, the AppUpdate plugin is able to download an
41      * available update. See Uploading Hybrid Apps in user documentation
42      * for information on how to upload an update to SAP Mobile
43      * Platform 3 server.
44      * <br/><br/>
45      * After an update is completely downloaded, the application
46      * user is
47      * prompted to install the update and restart the
48      * application. They can decline
49      * if they wish.
50      * <br/><br/>
51      * Once an update is installed, the application's revision
52      * number is updated.
53      * <br/><br/>
54      * <b>Adding and Removing the AppUpdate Plugin</b><br/>
55      * The AppUpdate plugin is added and removed using the
56      * <a href="http://cordova.apache.org/docs/en/edge/
57      * guide_cli_index.md.html#The%20Command-line%20Interface">Cordova
58      * CLI</a>.<br/>
59      * <br/>
60      * To add the AppUpdate plugin to your project, use the
61      * following command:<br/>
62      * cordova plugin add <path to directory containing Kapsel
63      * plugins>\appupdate<br/>
64      * <br/>
```

```
53      * To remove the AppUpdate plugin from your project, use the
following command:<br/>
54      * cordova plugin rm com.sap.mp.cordova.plugins.appupdate
55      * <br/><br/>
56      *
57      * <b>Hybrid App Revision Preference</b><br/>
58      * This is an optional preference that tells the AppUpdate
plugin if the local
59      * assets are uploaded to the server, and at what number. If
this preference is
60      * not provided, the default revision is 0.
61      * In your config.xml file you can add the following
preference:<br/>
62      * <preference name="hybridapprevision" value="1" />
63      <br/>
64      <br/>
65      * This means that the local assets in your www folder are
uploaded to the server
66      * and the server is reporting revision 1 for them. This
allows the application
67      * to receive a delta update when revision 2 is available,
instead of a full update.
68      * <br/><br/>
69      *
70      * <b>Caveats</b><br/>
71      * It is important to test that your update has valid HTML,
JavaScript, and CSS.
72      * Otherwise, the update could prevent the application from
functioning correctly,
73      * and may no longer be updateable. You can test the updated
application in a
74      * separate simulator or other test device. You can also
validate your
75      * JavaScript with tools like <a href="http://
www.jshint.com">JSHint</a>, or
76      * <a href="http://www.jshint.com">JSHint</a>.
```

Kapsel Development

```
77      * You can validate CSS with <a href="http://csslint.net">CSS
Lint</a>.
78      * <br/><br/>
79      *
80      * @namespace
81      * @alias AppUpdate
82      * @memberof sap
83      */
84      module.exports = {
85          /**
86           * Force an update check. By default, updates occur
automatically during logon and resume.
87           * See events for what is fired during this process.
88           * @example
89           * sap.AppUpdate.update();
90           */
91          update: function () {
92              // Abort if logon event has not yet fired
93              if (logonFired) {
94                  sap.Logon.unlock(function (connectionInfo) {
95                      //Add application ID required for REST call
96                      connectionInfo.applicationId =
sap.Logon.applicationId;
97
98                      exec(_eventHandler, null, 'AppUpdate',
'update', [connectionInfo]);
99                  });
100             }
101         },
102
103         /**
104          * Replaces the app resources with any newly downloaded
resources.
105          * @example
```

```
106     * sap.AppUpdate.reloadApp();
107     */
108     reloadApp: function () {
109         exec(null, null, 'AppUpdate', 'reloadApp', []);
110     },
111
112     /**
113     * Removes all local updates and loads the original Web
114     * assets bundled with the app. Call this after deleteRegistration.
115     * Reset calls error callback if it is called during the
116     * update process.
117     * @example
118     * sap.Logon.core.deleteRegistration(function() {
119     *     sap.AppUpdate.reset();
120     * }, function() {});
121     */
122     reset: function (successCallback, errorCallback) {
123         exec(successCallback, errorCallback, 'AppUpdate',
124             'reset', []);
125     },
126
127     /**
128     * Add a listener for an AppUpdate event. See events for
129     * available event names.
130     * @param {string} eventname Name of the app update
131     * event.
132     * @param {function} f Function to call when event is
133     * fired.
134     * @example
135     * sap.AppUpdate.addEventListener('checking', function(e)
136     {
137     *     console.log("Checking for update");
138     * });
139     */
140     addEventListener: function (eventname, f) {
```

Kapsel Development

```
134         if (eventname in channels) {
135             channels[eventname].subscribe(f);
136         }
137     },
138
139     /**
140     * Removes a listener for an AppUpdate event. See events
141     for available event names.
142     * @param {string} eventname Name of the app update
143     event.
144     * @param {function} f Function that was registered.
145     * @example
146     * // Adding the listener
147     * var listener = function(e) {
148     *     console.log("Checking for update");
149     * });
150     * sap.AppUpdate.addEventListener('checking',
151     listener);
152     * // Removing the listener
153     * sap.AppUpdate.removeEventListener('checking',
154     listener);
155     */
156     removeEventListener: function (eventname, f) {
157         if (eventname in channels) {
158             channels[eventname].unsubscribe(f);
159         }
160     }
161
162     /**
163     * Event fired when AppUpdate is checking for an
164     update.
165     *
166     * @event sap.AppUpdate#checking
```



```
163      * @type {object}
164      * @property {string} type - The name of the event. Value
is 'checking.'
165      * @example
166      * sap.AppUpdate.addEventListener('checking', function(e)
{
167          *     console.log("Checking for update");
168          * });
169      */
170
171     /**
172     * Event fired when AppUpdate finds no available updates
on the server.
173     *
174     * @event sap.AppUpdate#noupdate
175     * @type {object}
176     * @property {string} type - The name of the event. Value
is 'noupdate.'
177     * @example
178     * sap.AppUpdate.addEventListener('noupdate', function(e)
{
179         *     console.log("No update");
180         * });
181     */
182
183     /**
184     * Event fired when AppUpdate has found an update and is
starting the download.
185     *
186     * @event sap.AppUpdate#downloading
187     * @type {object}
188     * @property {string} type - The name of the event. Value
is 'downloading.'
189     * @example
```

```
190     * sap.AppUpdate.addEventListener('downloading',
function(e) {
191         *     console.log("Downloading update");
192     * });
193     */
194
195     /**
196     * Event fired when AppUpdate encounters an error while
checking for an update or while downloading an update.
197     * The status code and status message are provided with
this event.
198     *
199     * @event sap.AppUpdate#error
200     * @type {object}
201     * @property {string} type - The name of the event. Value
is 'error.'
202     * @property {int} statusCode - The HTTP error code.
203     * @property {string} statusMessage - The HTTP status
message.
204     * @example
205     * sap.AppUpdate.addEventListener('error', function(e)
{
206         *     console.log("Error downloading update. statusCode:
" + e.statusCode + " statusMessage: " + e.statusMessage);
207     * });
208     */
209
210     /**
211     * Event fired when AppUpdate has a newly downloaded
update available.
212     * A default handler is already added to
sap.AppUpdate.onupdateready that will ask the user to reload the
app.
213     * When using this event, you should call
sap.AppUpdate.reloadApp() to apply the downloaded update.
214     *
215     * @event sap.AppUpdate#updateready
```

```

216      * @type {object}
217      * @property {string} type - The name of the event. Value
is 'updateready.'
218      * @property {int} revision - The revision that was
downloaded.
219      * @example
220      *
221      * // This listens for updateready event.
222      * // Note: Use sap.AppUpdate.onupdateready if you want to
override the default handler.
223      * sap.AppUpdate.addEventListener('updateready',
function(e) {
224      *     console.log("Update ready");
225      * });
226      *
227      * // Override the default handler so that the update is
automatically loaded,
228      * // without first prompting the user for permission.
229      * sap.AppUpdate.onupdateready = function(e) {
230      *     // No notification just reload
231      *     console.log("Apply application update...");
232      *     sap.AppUpdate.reloadApp();
233      * };
234      *
235      * // Override the default handler with a custom prompt to
notify the user that the
236      * // application is ready to update.
237      * sap.AppUpdate.onupdateready = function() {
238      *     console.log("Confirming application updateâ€¦");
239      *     navigator.notification.confirm('Update
Available',
240      *         function(buttonIndex) {
241      *             if (buttonIndex === 2) {
242      *                 console.log("Applying application updateâ€¦");

```

Kapsel Development

```
243         *                 sap.AppUpdate.reloadApp();
244         *                 }
245         *                 },
246         *                 "Update", ["Later", "Relaunch Now"]);
247         * };
248         */
249     };
250
251     // Add getter/setter for DOM0 style events
252     for (var type in channels) {
253         function defineSetGet(eventType) {
254             module.exports.__defineGetter__("on" + eventType,
function () {
255                 return domZeroHandlers[eventType];
256             });
257
258             module.exports.__defineSetter__("on" + eventType,
function (val) {
259                 // Remove current handler
260                 if (domZeroHandlers[eventType]) {
261                     module.exports.removeEventListener(eventType,
domZeroHandlers[eventType]);
262                 }
263
264                 // Add new handler
265                 if (val) {
266                     domZeroHandlers[eventType] = val;
267                     module.exports.addEventListener(eventType,
domZeroHandlers[eventType]);
268                 }
269             });
270         }
271
272         defineSetGet(type);
```

```
273     }
274
275     // Add default update ready implementation
276     module.exports.onupdateready = function () {
277         if (!promptActive) {
278             promptActive = true;
279
280             var onConfirm = function (buttonIndex) {
281                 promptActive = false;
282                 if (buttonIndex === 2) {
283                     // Only reload if we are unlocked
284                     sap.Logon.unlock(function (connectionInfo)
285                     {
286                         //Add application ID required for REST
287                         call
288                         connectionInfo.applicationId =
289                         sap.Logon.applicationId;
290
291                         module.exports.reloadApp();
292                     });
293                 }
294             }
295
296             if (!bundle) {
297                 // Load required translations.
298                 var i18n =
299                 require('com.sap.mp.cordova.plugins.i18n.i18n');
300                 bundle = i18n.load({
301                     path: "plugins/
302                     com.sap.mp.cordova.plugins.appupdate/www"
```

```
302         bundle.get("update_available"),
303         onConfirm,
304         bundle.get("update"), [bundle.get("later"),
bundle.get("relaunch_now")]);
305     }
306 }
307
308 // When logon is ready, an update check is started.
309 document.addEventListener("onSapLogonSuccess", function ()
{
310     logonFired = true;
311     module.exports.update();
312 }, false);
313
314 document.addEventListener("onSapResumeSuccess",
module.exports.update, false);
```

Using the Logon Plugin

The Logon plugin is a component of SAP Mobile Application Framework (MAF) that is exposed as a Cordova plugin and provides an interface to the SAP Afaria client and Client Hub.

Note: Before implementing the Logon plugin, you should thoroughly understand the Client Hub service with which the plugin is integrated to enable onboarding. If you are using an iOS device, you must add the "clienthubEntitlements" to the Keychain Groups in the Entitlement section in Xcode.

Logon Plugin Overview

The Logon plugin manages the application registration and authentication processes through SAP Mobile Platform Server.

Most of the Kapsel plugins rely upon the services provided by the Logon plugin. This plugin manages the process of onboarding applications with SAP Mobile Platform Server, authenticating users, and so on. The Logon plugin, where available, interfaces with Client Hub and pulls certificates from Afaria.

The Logon plugin provides a login screen where the user can enter the values needed to connect to SAP Mobile Platform server, and which stores those values in its own secure data vault. This data vault is separate from the one that is provided with the EncryptedStorage plugin. The Logon plugin also lets the user lock and unlock the application, to protect sensitive data.

Adding the Logon Plugin

To install the Logon plugin, use the Cordova command line interface.

Prerequisites

- Set up the development environment.
- Create your Cordova Project.
- Add your OS platforms.

Task

1. Add the plugin, by entering, at the command prompt:

On Windows:

```
cordova -d plugin add <SDK_HOME>\MobileSDK3\KapselSDK
\plugins\logon
```

On Mac:

```
cordova -d plugin add ~<SDK_HOME>/MobileSDK3/KapselSDK/
plugins/logon
```

Note: The path you enter to the Kapsel plugin must be the absolute path (not relative path).

2. (Optional) To see a list of installed plugins in your Cordova project, open a command prompt or terminal window, navigate to your Cordova project folder, and enter:

```
cordova plugins
```

The Cordova command line interface returns a JSON array showing installed plugins, for example:

```
[ 'com.sap.mp.cordova.plugins.corelibs',
'com.sap.mp.cordova.plugins.logon',
'org.apache.cordova.console',
'org.apache.cordova.device',
'org.apache.cordova.device-orientation',
'org.apache.cordova.dialogs',
'org.apache.cordova.inappbrowser' ]
```

In this case, some core Cordova plugins were added, including corelibs, console, device, device-orientation, dialogs, and inAppBrowser. CoreLibs is a utility plugin that is automatically added to every Kapsel project by the command line interface, so you need never add the CoreLibs plugin to a project manually.

3. Configure the application in Management Cockpit.
4. Define a variable in the JavaScript code (typically, this is done in the `index.html` file of your Cordova application) to describe the app ID, for example:

```
var appId = "com.sap.kapsel.mykapselapp";
```

Kapsel uses an app ID to tell the server which application definition on the server to use for this application. The app ID that is defined on the server must match what is entered here.

5. Define the connection to the server, for example:

```
var defaultContext = {
    "serverHost" : "192.168.254.159",
    "https" : "false",
    "serverPort" : "8080",
};
```

This prepopulates the fields in the registration dialog that is shown to users during the initialization process.

6. Make a call to the Logon plugin's `init` method as shown:

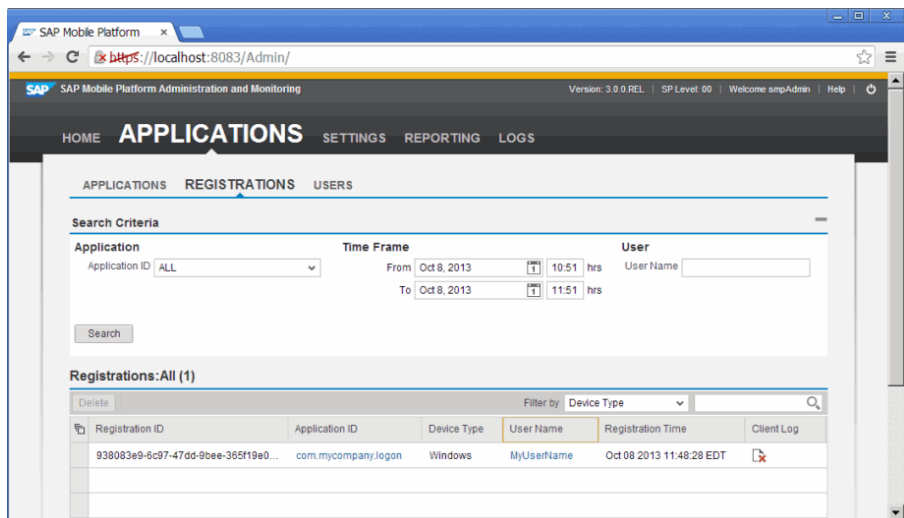
```
//Make call to Logon's Init method to get things registered and
all set up
sap.Logon.init(logonSuccess, logonError, appId, defaultContext);
```

The `init` method gathers information about the environment's security configuration by asking the Afaria client and Client Hub application, if available, sets up and configures the DataVault, connects to the server to register the application connection and authenticate the user. As part of this process, the appropriate screens are shown to gather user input and manage the entire process.

7. Verify the registration in Management Cockpit.

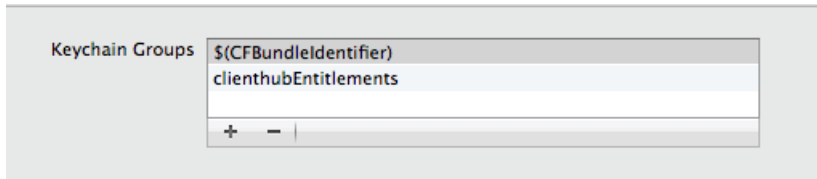
- a) Log in to Management Cockpit.
- b) Click **Applications**.
- c) Click **Registrations**.

You can see the registration ID following a successful registration.



8. Use the Android IDE or Xcode to deploy and run the project.

Note: If you are deploying to an iOS device, in Xcode, you must add the `clienthubEntitlements` and `$(CFBundleIdentifier)` to the keychain group in the Entitlements section as well as the bundle identifier.



Configuring Default Values

Add JavaScript to configure default logon settings.

1. Go to the `<Project Name>/www` folder and open the file where you want to add the JavaScript, for example, `index.html`.
2. Add your code, for example:

```
function logonSuccessCallback(context) {
    console.log("logonSuccessCallback " +
JSON.stringify(context));
}

function errorCallback(e) {
    alert("An error occurred");
    alert(JSON.stringify(e));
}

function deviceReady() {

    var appId = "theAppId"; // Change this to app id on
server

    // Optional initial connection context
    var context = {
        "serverHost": "example.com",
        "https": "false",
        "serverPort": "8080",
        "communicatorId": "REST",
    };
    sap.Logon.init(logonSuccessCallback, errorCallback,
appId, context);
}

document.addEventListener("deviceready", deviceReady,
false);
```

This example shows the call to the `sap.Logon.init` function, as well as the success and error callbacks that are passed to the `sap.Logon.init` function. It also shows how you can make sure the registration process is started as soon as possible by attaching a

listener to the `deviceready` event. Inside the `deviceReady` function, the app ID and the context are defined.

3. Save the file.

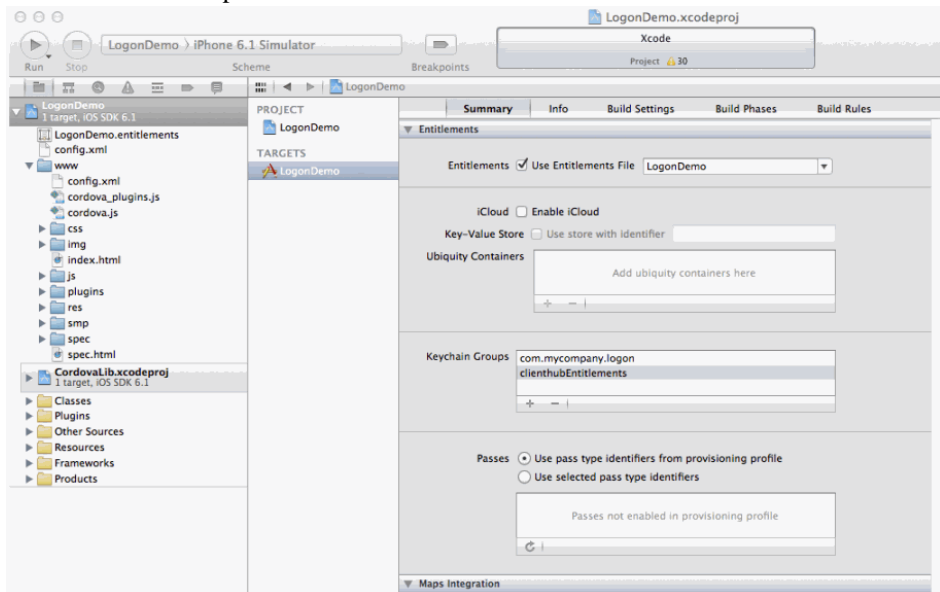
Running the Logon Application on iOS

Deploy and run the Logon project on iOS.

1. In a terminal window, make sure you are in the project folder and execute the command:

```
cordova prepare ios
```
2. Open Xcode.
3. In a Finder window, browse to your Cordova project folder, `<Project Name>/platforms/ios`.
4. Add the **clienthubEntitlements** keychain group to the Entitlements section of the project.

This shows an example:



5. Double-click the `<ProjectName>.xcodeproj` file to open the project in Xcode.
6. Select your Simulator type and click the **Run** button.

Removing Fields From the Registration Screen

If your application does not use a relay server, a reverse proxy server, or connect to an SAP Mobile Platform 2.x server, you can remove some of the fields from the registration screen, such as the URL Suffix, Company ID, and Security Config.

1. Open the `StaticScreens.js` file, which is located in `SDK_HOME \MobileSDK3\KapselSDK\plugins\logon\www\common\modules`.
2. Find the `SCR_REGISTRATION` screen and reorder, or hide and show fields using the `visible:false` options. You can also delete unneeded entries.

For example:

```
SCR_REGISTRATION': {
  id: 'SCR_REGISTRATION',
  fields: {
    user : {
      uiKey:'FLD_USER'
    },
    password : {
      uiKey:'FLD_PASS',
      type: 'password'
    },
    serverHost : {
      uiKey:'FLD_HOST',
      editable:true
    },
    serverPort : {
      uiKey:'FLD_PORT',
      type: 'number',
      editable:true,
      visible:true
    },
    communicatorId : {
      uiKey: 'FLD_COMMUNICATORID',
      'default':'REST',
      visible:false
    },
    https: {
      uiKey:'FLD_IS_HTTPS',
      type: 'switch',
      'default':false,
      visible:false
    },
  },
},
```

3. Save the file.

Kapsel Logon API Reference

The Kapsel Logon API Reference provides usage information for Logon API classes and methods, as well as provides sample source code.

Logon namespace

The Logon plugin provides screen flows to register an app with an SAP Mobile Platform server.

The logon plugin is a component of the SAP Mobile Application Framework (MAF), exposed as a Cordova plugin. The basic idea is that it provides screen flows where the user can enter the values needed to connect to an SAP Mobile Platform 3.0 server and stores those values in its own secure data vault. This data vault is separate from the one provided with the encrypted storage plugin. In an OData based SAP Mobile Platform 3.0 application, a client must onboard or register with the SAP Mobile Platform 3.0 server to receive an application connection ID for a particular app. The application connection ID must be sent along with each request that is proxied through the SAP Mobile Platform 3.0 server to the OData producer.

Adding and Removing the Logon Plugin

The Logon plugin is added and removed using the *Cordova CLI*.

To add the Logon plugin to your project, use the following command:

```
cordova plugin add <full path to directory containing Kapsel plugins>\logon
```

To remove the Logon plugin from your project, use the following command:

```
cordova plugin rm com.sap.mp.cordova.plugins.logon
```

Methods

Name	Description
<i>changePassword(onSuccess, onerror)</i> on page 51	This method will launch the UI screen for application users to manage and update the back end passcode that Logon stores in the data vault that is used to authenticate the client to the server.
<i>get(onSuccess, onerror, key)</i> on page 51	Get an (JSON serializable) object from the DataVault for a given key.
<i>init(successCallback, errorCallback, applicationId, [context], [logon View])</i> on page 52	Initialization method to set up the Logon plugin.
<i>lock(onSuccess, onerror)</i> on page 56	Locks the Logon plugin's secure data vault.

<i>managePasscode(onSuccess, onerror)</i> on page 56	This method will launch the UI screen for application users to manage and update the data vault passcode or, if the SMP server's Client Passcode Policy allows it, enable or disable the passcode to the data vault.
<i>set(onSuccess, onerror, key, value)</i> on page 57	Set an (JSON serializable) object in the Data-Vault.
<i>showRegistrationData(onSuccess, onerror)</i> on page 58	Calling this method will show a UI screen with values used for registering application.
<i>unlock(onSuccess, onerror)</i> on page 59	

Type Definitions

Name	Description
<i>errorCallback(errorObject)</i> on page 59	Callback function that is invoked in case of an error.
<i>getSuccessCallback(value)</i> on page 60	Callback function that is invoked upon successfully retrieving an object from the DataVault.
<i>successCallback(context)</i> on page 60	Callback function that is invoked upon successfully registering or unlocking or retrieving the context.
<i>successCallbackNoParameters</i> on page 65	Callback function that will be invoked with no parameters.

Source

LogonController.js, line 1223 on page 108.

applicationId member

The application ID with which *sap.Logon.init* on page 52 was called. It is available here so it is easy to access later.

Syntax

```
<static> applicationId
```

Example

```
// After calling the init function
alert("The app ID for this app is: " + sap.Logon.applicationId);
```

Source

LogonController.js, line 1281 on page 110.

core member

Direct reference to the logon core object used by the Logon plugin.

This is needed to perform more complex operations that are not generally needed by applications.

There are several functions that can be accessed on the core object:

`getState(successCallback,errorCallback)` returns the state object of the application to the success callback in the form of a JavaScript object.

`getContext(successCallback,errorCallback)` returns the context object of the application to the success callback in the form of a JavaScript object.

`deleteRegistration(successCallback,errorCallback)` deletes the application's registration from the SAP Mobile Platform server and removes

application data on device.

Syntax

`<static> core`

Example

```
var successCallback = function(result) {
    alert("Result: " + JSON.stringify(result));
}
var errorCallback = function(errorInfo) {
    alert("Error: " + JSON.stringify(errorInfo));
}
sap.Logon.core.getState(successCallback,errorCallback);
sap.Logon.core.getContext(successCallback,errorCallback);
sap.Logon.core.deleteRegistration(successCallback,errorCallback);
```

Source

LogonController.js, line 1301 on page 111.

changePassword(onSuccess, onerror) method

This method will launch the UI screen for application users to manage and update the back end passcode that Logon stores in the data vault that is used to authenticate the client to the server.

Syntax

```
<static> changePassword( onSuccess, onerror )
```

Parameters

Name	Type	Description
<i>onsuccess</i>	<i>sap.Logon~successCallback-NoParameters</i> on page 65	The callback to call if the screen flow succeeds. onsuccess will be called without parameters for this method.
<i>onerror</i>	<i>sap.Logon~errorCallback</i> on page 59	The function that is invoked in case of an error.

Example

```
var errorCallback = function(errorInfo) {
    alert("Error: " + JSON.stringify(errorInfo));
}
var successCallback = function(context) {
    alert("Password successfully changed.");
}
sap.Logon.changePassword(successCallback, errorCallback);
```

Source

LogonController.js, line 1455 on page 117.

get(onSuccess, onerror, key) method

Get an (JSON serializable) object from the DataVault for a given key.

Syntax

```
<static> get( onSuccess, onerror, key )
```

Parameters

Name	Type	Description
------	------	-------------

<i>onsuccess</i>	<i>sap.Logon~getSuccessCallback</i> on page 60	The function that is invoked upon success. It is called with the resulting object as a single parameter. This can be null or undefined, if no object is defined for the given key.
<i>onerror</i>	<i>sap.Logon~errorCallback</i> on page 59	The function to invoke in case of error.
<i>key</i>	string	The key with which to query the DataVault.

Example

```
var errorCallback = function(errorInfo) {
    alert("Error: " + JSON.stringify(errorInfo));
}
var getSuccess = function(value) {
    alert("value retrieved from the store: " +
JSON.stringify(value));
}
var setSuccess = function() {
    sap.Logon.get(getSuccess, errorCallback, 'someKey');
}
sap.Logon.set(setSuccess, errorCallback, 'someKey', 'some string
(could also be an object).');
```

Source

LogonController.js, line 1323 on page 112.

init(successCallback, errorCallback, applicationId, [context], [logonView]) method
Initialization method to set up the Logon plugin.

This will register the application with the SMP server and also authenticate the user with servers on the network. This step must be done first prior to any attempt to communicate with the SMP server.

Syntax

<static> *init(successCallback, errorCallback, applicationId, [context], [logonView])*

Parameters

Name	Type	Argument	Default	Description
------	------	----------	---------	-------------

<i>successCallback</i>	<i>sap.Logon~successCallback</i> on page 60			The function that is invoked if initialization is successful. The current context is passed to this function as the parameter.
<i>errorCallback</i>	<i>sap.Logon~errorCallback</i> on page 59			The function that is invoked in case of an error.
<i>applicationId</i>	string			The unique ID of the application. Must match the application ID on the SAP Mobile Platform server.

<i>context</i>	object	(optional)		<p>The context with default values for application registration. See <i>sap.Logon~successCallback</i> on page 60 for the structure of the context object. Note that all properties of the context object are optional, and you only need to specify the properties for which you want to provide default values for. The values will be presented to the application users during the registration process and given them a chance to override these values during runtime.</p>
----------------	--------	------------	--	---

<i>logonView</i>	string	(optional)	"com/sap/mp/logon/iabui"	The cordova module ID of a custom renderer for the logon, implementing the [showScreen(), close()] interface. Please use the default module unless you are absolutely sure that you can provide your own custom implementation. Please refer to JavaScript files inside your Kapsel project's plugins\logon\www\common\modules\ folder as example.
------------------	--------	------------	--------------------------	--

Example

```
// a custom UI can be loaded here
var logonView = sap.logon.IabUi;

// The app ID
var applicationId = "someAppID";

// You only need to specify the fields for which you want to set the
// default. These values are optional because they will be
// used to prefill the fields on Logon's UI screen.
var defaultContext = {
  "serverHost" : "defaultServerHost.com"
  \t"https" : false,
  \t"serverPort" : "8080",
  \t"user" : "user1",
  \t"password" : "Zzzzzz123",
  \t"communicatorId" : "REST",
  \t"securityConfig" : "sec1",
  \t"passcode" : "Aaaaaa123",
  \t"unlockPasscode" : "Aaaaaa123"
};

var app_context;

var successCallback = function(context) {
```

```
    app_context = context;
  }

  var errorCallback = function(errorInfo) {
    alert("error: " + JSON.stringify(errorInfo));
  }
  sap.Logon.init(successCallback, errorCallback, applicationId,
  defaultContext, logonView);
```

Source

LogonController.js, line 1273 on page 110.

lock(onSuccess, onerror) method

Locks the Logon plugin's secure data vault.

Syntax

<static> lock(*onSuccess*, *onerror*)

Parameters

Name	Type	Description
<i>onSuccess</i>	<i>sap.Logon~successCallback-NoParameters</i> on page 65	The function to invoke upon success.
<i>onerror</i>	<i>sap.Logon~errorCallback</i> on page 59	The function to invoke in case of error.

Example

```
var errorCallback = function(errorInfo) {
  alert("Error: " + JSON.stringify(errorInfo));
}
var successCallback = function() {
  alert("Locked!");
}
sap.Logon.lock(successCallback, errorCallback);
```

Source

LogonController.js, line 1362 on page 113.

managePasscode(onSuccess, onerror) method

This method will launch the UI screen for application users to manage and update the data vault passcode or, if the SMP server's Client Passcode Policy allows it, enable or disable the passcode to the data vault.

Syntax

<static> managePasscode(*onSuccess*, *onerror*)

Parameters

Name	Type	Description
<i>onsuccess</i>	<i>sap.Logon~successCallback-NoParameters</i> on page 65	The function to invoke upon success.
<i>onerror</i>	<i>sap.Logon~errorCallback</i> on page 59	The function to invoke in case of error.

Example

```
var errorCallback = function(errorInfo) {
    alert("Error: " + JSON.stringify(errorInfo));
}
var successCallback = function(context) {
    alert("Passcode successfully managed.");
}
sap.Logon.managePasscode(successCallback, errorCallback);
```

Source

LogonController.js, line 1436 on page 116.

set(onsuccess, onerror, key, value) method

Set an (JSON serializable) object in the DataVault.

Syntax

<static> *set(onsuccess, onerror, key, value)*

Parameters

Name	Type	Description
<i>onsuccess</i>	<i>sap.Logon~successCallback-NoParameters</i> on page 65	The function to invoke upon success. <i>onsuccess</i> will be called without parameters for this method.
<i>onerror</i>	<i>sap.Logon~errorCallback</i> on page 59	The function to invoke in case of error.
<i>key</i>	string	The key to store the provided object on.
<i>value</i>	object	The object to be set on the given key. Must be JSON serializable (ie: cannot contain circular references).

Example

```
var errorCallback = function(errorInfo) {
    alert("Error: " + JSON.stringify(errorInfo));
}
var getSuccess = function(value) {
    alert("value retrieved from the store: " +
JSON.stringify(value));
}
var setSuccess = function() {
    sap.Logon.get(getSuccess, errorCallback, 'someKey');
}
sap.Logon.set(setSuccess, errorCallback, 'someKey', 'some string
(could also be an object).');
```

Source

LogonController.js, line 1346 on page 113.

showRegistrationData(onSuccess, onerror) method

Calling this method will show a UI screen with values used for registering application.

Syntax

<static> showRegistrationData(*onSuccess*, *onerror*)

Parameters

Name	Type	Description
<i>onSuccess</i>	<i>sap.Logon~successCallback-NoParameters</i> on page 65	The callback to call if the screen flow succeeds. onSuccess will be called without parameters for this method.
<i>onerror</i>	<i>sap.Logon~errorCallback</i> on page 59	The function that is invoked in case of an error.

Example

```
var errorCallback = function(errorInfo) {
    alert("Error: " + JSON.stringify(errorInfo));
}
var successCallback = function(context) {
    alert("The showRegistrationData screenflow was successful.");
}
sap.Logon.showRegistrationData(successCallback, errorCallback);
```

Source

LogonController.js, line 1472 on page 118.

unlock(onSuccess, onError) method

Unlock the Logon plugin's secure data vault if it has been locked (due to being inactive, or *sap.Logon.lock* on page 56 being called), then the user is prompted for the passcode to unlock the application.

If the application is already unlocked, then nothing will be done.

If the application has passcode disabled, then passcode prompt will not be necessary. In all cases if an error does not occur, the success callback is invoked with the current logon context as the parameter.

Syntax

```
<static> unlock( onSuccess, onError )
```

Parameters

Name	Type	Description
<i>onsuccess</i>	<i>sap.Logon~successCallback</i> on page 60	The callback to call if the screen flow succeeds. <i>onsuccess</i> will be called with the current logon context as a single parameter.
<i>onerror</i>	<i>sap.Logon~errorCallback</i> on page 59	The callback to call if the screen flow fails.

Example

```
var errorCallback = function(errorInfo) {
    alert("Error: " + JSON.stringify(errorInfo));
}
var successCallback = function(context) {
    alert("Registered and unlocked. Context: " +
JSON.stringify(context));
}
sap.Logon.unlock(successCallback, errorCallback);
```

Source

LogonController.js, line 1385 on page 114.

errorCallback(errorObject) type

Callback function that is invoked in case of an error.

Syntax

`errorCallback(errorObject)`

Parameters

Name	Type	Description
<i>errorObject</i>	Object	An object containing properties: 'errorCode', 'errorMessage', and 'errorDomain'. The 'errorCode' is just a number uniquely identifying the error. The 'errorMessage' property is a string with more detailed information of what went wrong. The 'errorDomain' property specifies the domain that the error occurred in.

Source

LogonController.js, line 1476 on page 118.

getSuccessCallback(value) type

Callback function that is invoked upon successfully retrieving an object from the Data Vault.

Syntax

`getSuccessCallback(value)`

Parameters

Name	Type	Description
<i>value</i>	Object	The object that was stored with the given key. Can be null or undefined if no object was stored with the given key.

Source

LogonController.js, line 1482 on page 118.

successCallback(context) type

Callback function that is invoked upon successfully registering or unlocking or retrieving the context.

Syntax`successCallback(context)`**Parameters**

Name	Type	Description
------	------	-------------

<p><i>context</i></p>	<p>Object</p>	<p>An object containing the current logon context. Two properties of particular importance are applicationEndpointURL, and applicationConnectionId. The context object contains the following properties:</p> <p>"registrationContext": {</p> <p> "serverHost": Host of the server.</p> <p> "domain": Domain for server. Can be used in case of SAP Mobile Platform communication.</p> <p> "resourcePath": Resource path on the server. The path is used mainly for path based reverse proxy but can contain a custom relay server path as well.</p> <p> "https": Marks whether the server should be accessed in a secure way.</p> <p> "serverPort": Port of the server.</p> <p> "user": Username in the backend.</p> <p> "password": Password for the backend user.</p>
-----------------------	---------------	---

"farmId": FarmId of the server. Can be nil. Used in case of Relay server or SiteMinder.

"communicatorId": Id of the communicator manager that will be used for performing the logon. Possible values: IMO / GATEWAY / REST

"securityConfig": Security configuration. If nil, the default configuration is used.

"mobileUser": Mobile User. Used in case of IMO manual user creation.

"activationCode": Activation Code. Used in case of IMO manual user creation.

"gatewayClient": The key string that identifies the client on the gateway. Used in Gateway only registration mode. The value will be used as adding the parameter: sap-client=<gateway client>

"gatewayPingPath": The custom path of the ping URL on the gateway. Used in case of Gateway only registration mode.

}

		<p>"applicationEndpointURL": Contains the application endpoint URL after a successful registration.</p>
		<p>"applicationConnectionId": ID to get after a successful SUP REST registration. Needs to be set in the download request header with key X-SUP-APP-CID</p>
		<p>"afariaRegistration": manual / automatic / certificate</p>
		<p>"policyContext": Contains the password policy for the secure store {</p>
		<p>"alwaysOn":</p>
		<p>"alwaysOff":</p>
		<p>"defaultOn":</p>
		<p>"hasDigits":</p>
		<p>"hasLowerCaseLetters":</p>
		<p>"hasSpecialLetters":</p>
		<p>"hasUpperCaseLetters":</p>
		<p>"defaultAllowed":</p>

		<pre>"expirationDays":</pre>
		<pre>"lockTimeout":</pre>
		<pre>"minLength":</pre>
		<pre>"minUniqueChars":</pre>
		<pre>"retryLimit":</pre>
		<pre>}</pre>
		<pre>"registrationReadOnly": specifies whether context values are coming from clientHub / afaria</pre>
		<pre>"policyReadOnly": specifies whether passcode policy is coming from afaria</pre>
		<pre>"credentialsByClientHub": specifies whether credentials are coming from clientHub</pre>

Source

LogonController.js, line 1478 on page 118.

successCallbackNoParameters type

Callback function that will be invoked with no parameters.

Syntax

```
successCallbackNoParameters()
```

Source

LogonController.js, line 1480 on page 118.

Source code

LogonController.js

```
1
2
3     var utils = sap.logon.Utills;
4     var TIMEOUT = 2000;
5
6     var _oLogonCore;
7     var _oLogonView;
8     var _hasLogonSuccessEventFired = false;
9
10    var _providedContext;
11
12    var init = function (successCallback, errorCallback,
13    applicationId, context, customView) {
14
15        document.addEventListener("resume",
16            function() {
17                resume (
18                    function() { fireEvent('onSapResumeSuccess',
19    arguments);},
20                    function() { fireEvent('onSapResumeError',
21    arguments);}
22                );
23            },
24            false);
25
26    // The success callback used for the call to
27    _oLogonCore.initLogon(...)
28    var initSuccess = function(){
29        utils.log('LogonController: LogonCore successfully
30    initialized.');
```

```
28         // Now that Logon is initialized, registerOrUnlock
is automatically called.
29         registerOrUnlock( successCallback,
errorCallback );
30     }
31
32     var initError = function(error){
33         // If a parameter describing the error is given,
pass it along.
34         // Otherwise, construct something to call the error
callback with.
35         if( error ) {
36             errorCallback( error );
37         } else {
38             errorCallback( utils.Error('ERR_INIT_FAILED') );
39         }
40     }
41
42     utils.log('LogonController.init enter');
43     utils.log(applicationId);
44     module.exports.applicationId = applicationId;
45
46     // Make note of the context given (if any)
47     if( context ){
48         _providedContext = context;
49     }
50
51     _oLogonView = customView;
52     if (!_oLogonView) {
53         _oLogonView = sap.logon.IabUi;
54     }
55
```

Kapsel Development

```
56          //
coLogonCore.cordova.require("com.sap.mp.cordova.plugins.logon.Logon
Core");

57          _oLogonCore = sap.logon.Core;

58          _oLogonCore.initLogon(initSuccess, initError,
applicationId);

59

60          //update exports definition
61          module.exports.core = _oLogonCore;
62      }
63
64      var fireEvent = function (eventId, args) {
65          if (typeof eventId === 'string') {
66              //var event = document.createEvent('Events');
67              //event.initEvent(eventId, false, false);
68
69              if (!window.CustomEvent) {
70                  window.CustomEvent = function(type,
eventInitDict) {
71                      var newEvent =
document.createEvent('CustomEvent');
72                      newEvent.initCustomEvent(
73                          type,
74                          !(eventInitDict &&
eventInitDict.bubbles),
75                          !(eventInitDict &&
eventInitDict.cancelable),
76                          (eventInitDict ? eventInitDict.detail :
null));
77                      return newEvent;
78                  };
79              }
80
81          var event = new CustomEvent(eventId, { 'detail':
{ 'id': eventId, 'args': args }});
82
```



```
83         setTimeout(function() {
84             document.dispatchEvent(event);
85         }, 0);
86     } else {
87         throw 'Invalid eventId: ' +
JSON.stringify(event);
88     }
89 }
90
91     var FlowRunner = function(onsuccess, onerror, pLogonView,
pLogonCore) {
92
93         var onFlowSuccess;
94         var onFlowError;
95         var onFlowCancel;
96
97         var logonView;
98         var logonCore;
99         var flow;
100
101
102
103         logonView = pLogonView;
104         logonCore = pLogonCore;
105
106         onFlowSuccess = function onFlowSuccess() {
107             utils.logJSON('onFlowSuccess');
108             logonView.close();
109             onsuccess.apply(this, arguments);
110         }
111
112         onFlowError = function onFlowError() {
113             utils.logJSON('onFlowError');
```

```
114         logonView.close();
115         onerror.apply(this, arguments);
116     }
117
118     onFlowCancel = function onFlowCancel(){
119         utils.logJSON('onFlowCancel');
120         //logonView.close();
121         onFlowError(new
utils.Error('ERR_USER_CANCELLED'));
122     }
123
124     var handleCoreStateOnly = function(currentState){
125         handleCoreResult(null, currentState);
126     }
127
128     var handleCoreResult = function (currentContext,
currentState) {
129         if (typeof currentContext === undefined)
currentContext = null;
130
131         //workaround for defaultPasscodeAllowed
132         if (currentState) {
133             if (currentContext &&
currentContext.policyContext &&
currentContext.policyContext.defaultAllowed){
134                 currentState.defaultPasscodeAllowed =
true;
135             }
136             else {
137                 currentState.defaultPasscodeAllowed =
false;
138             }
139         }
140
```

```
141         utils.logJSON(currentContext, 'handleCoreResult
currentContext');
142         utils.logJSON(currentState, 'handleCoreResult
currentState');
143
144
145         utils.logJSON(flow.name);
146         var matchFound = false;
147         var rules = flow.stateTransitions;
148
149
150         ruleMatching:
151         for (key in rules){
152
153             var rule = flow.stateTransitions[key];
154             //utils.logJSON(rule, 'rule');
155
156             //utils.logJSON(rule.condition,
'rule.condition');
157             if (typeof rule.condition === 'undefined')
{
158                 throw 'undefined condition in state
transition rule';
159             }
160
161
162             if (rule.condition.state === null) {
163                 if (currentState)
164                 {
165                     continue ruleMatching; // non-null state
(and rule) mismatch
166                 }
167                 //else {
168                 //    // match:
```

Kapsel Development

```
169 // // rule.condition.state === null
170 // // (typeof currentState ===
'undefined') // null or undefined
171 //}
172 }
173 else if (rule.condition.state !== 'undefined' &&
currentState){
174     utils.log('stateMatching');
175
176     stateMatching:
177     for (field in rule.condition.state) {
178         utils.log(field);
179         if (rule.condition.state[field] ===
currentState[field])
180             {
181                 utils.log('field matching ' +
field);
182                 continue stateMatching; // state
field match
183             }
184         else {
185             utils.log('field mismatching ' +
field);
186             continue ruleMatching; // state field
(and rule) mismatch
187         };
188     }
189 }
190
191 if (rule.condition.context === null) {
192     if (currentContext)
193     {
194         continue ruleMatching; // non-null
context (and rule) mismatch
195     }
```

```

196                                     //else {
197                                     //    // match:
198                                     //    // rule.condition.context === null
198                                     &&
199                                     //    // (typeof currentContext ===
199                                     'undefined') // null or undefined
200                                     //}
201                                     }
202     else if (rule.condition.context !== 'undefined'
202     && currentContext){
203
204         utils.log('contextMatching');
205         contextMatching:
206         for (field in rule.condition.context) {
207             utils.log(field);
208             if (rule.condition.context[field] ===
208             currentContext[field])
209                 {
210                     utils.log('field matching ' +
210                     field);
211                     continue contextMatching; // context
211                     field match
212                 }
213                 else {
214                     utils.log('field mismatching ' +
214                     field);
215                     continue ruleMatching; // context
215                     field (and rule) mismatch
216                 };
217             }
218         }
219         utils.log('match found');
220         utils.logJSON(rule, 'rule');
221
222         if (typeof rule.action === 'function') {

```

```
223             rule.action(currentContext);
224         }
225         else if (typeof rule.action === 'string') {
226             // the action is a screenId
227             var screenId = rule.action;
228             utils.log('handleCoreResult: ' +
229 screenId);
229         utils.logKeys(flow.screenEvents[screenId]);
230             if(!currentContext){
231                 currentContext = {};
232             }
233
234             if( !currentContext.registrationContext &&
235 _providedContext ){
236                 // The current registrationContext is
237 null, and we have been given a context when initialized,
238                 // so use the one we were given.
239                 currentContext.registrationContext =
240 _providedContext;
241             } else if
242 (currentContext.registrationContext && _providedContext && !
243 currentContext.registrationReadOnly){
244                 for (key in _providedContext) {
245                     //if (!
246 currentContext.registrationContext[key]){
247
248                 currentContext.registrationContext[key] = _providedContext[key];
249             }
250             }
251         }
252
253         logonView.showScreen(screenId,
254 flow.screenEvents[screenId], currentContext);
255
256     }
257 }
```

```
249         else {
250             onFlowError(new
utils.Error('ERR_INVALID_ACTION'));
251         }
252
253         matchFound = true;
254         break ruleMatching;
255     }
256
257     if (!matchFound) {
258         onFlowError(new
utils.Error('ERR_INVALID_STATE'));
259     }
260 }
261
262
263     this.run = function(FlowClass) {
264         utils.log('FlowRunner.run ' + FlowClass.name);
265         flow = new FlowClass(logonCore, logonView,
handleCoreResult, onFlowSuccess, onFlowError, onFlowCancel);
266         utils.logKeys(flow , 'new flow ');
267         logonCore.getState(handleCoreStateOnly,
onFlowError);
268     }
269
270 }
271
272
273     var MockFlow = function MockFlow(logonCore, logonView,
onCoreResult, onFlowSuccess, onFlowError, onFlowCancel) {
274         //wrapped into a function to defer evaluation of the
references to flow callbacks
275         //var flow = {};
276
```

```
277         this.name = 'mockFlowBuilder';
278
279         this.stateTransitions = [
280             {
281                 condition: {
282                     state: {
283                         secureStoreOpen: false,
284                     }
285                 },
286                 action: 'SCR MOCKSCREEN'
287             },
288             {
289                 condition: {
290                     state: {
291                         secureStoreOpen: true,
292                     }
293                 },
294                 action: 'SCR MOCKSCREEN'
295             },
296
297         ];
298
299         this.screenEvents = {
300             'SCR_TURN_PASSCODE_ON': {
301                 onsubmit: onFlowSuccess,
302                 oncancel: onFlowCancel,
303                 onerror: onFlowError,
304             }
305         };
306
307         utils.log('flow constructor return');
308         //return flow;
```



```
309         }
310
311         var RegistrationFlow = function
RegistrationFlow(logonCore, logonView, onCoreResult, onFlowSuccess,
onFlowError, onFlowCancel) {
312             //wrapped into a function to defer evaluation of the
references to flow callbacks
313
314             this.name = 'registrationFlowBuilder';
315
316             var registrationInProgress = false;
317
318             var onCancelSSOPin = function() {
319                 onFlowError(errorWithDomainCodeDescription("MAFLogon","0","SSO
Passcode set screen was cancelled"));
320             }
321
322             var onCancelRegistration = function() {
323                 onFlowError(errorWithDomainCodeDescription("MAFLogon","1","Registra
tion screen was cancelled"));
324             }
325
326             // internal methods
327             var showScreen = function(screenId) {
328                 return function(coreContext) {
329                     logonView.showScreen(screenId,
this.screenEvents[screenId], coreContext);
330                 }.bind(this);
331             }.bind(this);
332
333             var onUnlockSubmit = function(context){
334                 utils.logJSON(context,
'logonCore.unlockSecureStore');
```

```
335         logonCore.unlockSecureStore (onCoreResult,
onUnlockError, context)
336     }
337
338     var onUnlockError = function(error) {
339         utils.logJSON("onUnlockError: " +
JSON.stringify(error));
340
341         // TODO switch case according to the error
codes
342         logonView.showNotification("ERR_UNLOCK_FAILED");
343     }
344
345     var noOp = function() { }
346
347     var onRegistrationBackButton = function() {
348         if (registrationInProgress == true) {
349             utils.log('back button pushed, no operation
is required as registration is running');
350         }
351         else {
352             onCancelRegistration();
353         }
354     }
355
356     var onUnlockVaultWithDefaultPasscode = function()
{
357         utils.log('logonCore.unlockSecureStore -
default passcode');
358         var unlockContext =
{"unlockPasscode":null};
359         logonCore.unlockSecureStore (onCoreResult,
onFlowError, unlockContext)
360     }
361
```

```
362         var onRegSucceeded = function(context, state) {
363             onCoreResult(context, state);
364             registrationInProgress = false;
365         }
366
367         var onRegError = function(error){
368             utils.logJSON(error, 'registration
failed');
369             logonView.showNotification(getRegistrationErrorText(error));
370             registrationInProgress = false;
371         }
372
373         var onRegSubmit = function(context){
374             utils.logJSON(context,
'logonCore.startRegistration');
375             registrationInProgress = true;
376             logonCore.startRegistration(onRegSucceeded,
onRegError, context)
377         }
378
379         var onCreatePasscodeSubmit = function(context){
380             utils.logJSON(context,
'logonCore.persistRegistration');
381             logonCore.persistRegistration(onCoreResult,
onCreatePasscodeError, context);
382         }
383
384         var onCancelRegistrationError = function(error)
{
385             utils.logJSON("onCancelRegistrationError: " +
JSON.stringify(error));
386             logonView.showNotification(getRegistrationCancelError(error));
387         }
388
```

Kapsel Development

```
389         var onCreatePasscodeError = function(error) {
390             utils.logJSON("onCreatePasscodeError: " +
JSON.stringify(error));
391         logonView.showNotification(getSecureStoreErrorText(error));
392     }
393
394     var onSSOPasscodeSetError = function(error) {
395         utils.logJSON("onSSOPasscodeSetError: " +
JSON.stringify(error));
396         logonView.showNotification(getSSOPasscodeSetErrorText(error));
397     }
398
399     var callGetContext = function(){
400         utils.log('logonCore.getContext');
401         logonCore.getContext(onCoreResult,
onFlowError);
402     }
403
404     var onFullRegistered = function()
405     {
406         var getContextSuccessCallback =
function(result){
407
408             if(!_hasLogonSuccessEventFired) {
409                 fireEvent("onSapLogonSuccess",
arguments);
410                 _hasLogonSuccessEventFired = true;
411             }
412
413             onFlowSuccess(result);
414         }
415         utils.log('logonCore.getContext');
```

```
416 logonCore.getContext(getContextSuccessCallback, onFlowError);
417     }
418
419     var onForgotAppPasscode = function(){
420         utils.log('logonCore.deleteRegistration');
421         logonCore.deleteRegistration(onFlowError,
422 onFlowError);
422     }
423
424     var onForgotSsoPin = function(){
425         utils.log('forgotSSOPin');
426
427 logonView.showNotification("ERR_FORGOT_SSO_PIN");
427     }
428
429     var onSkipSsoPin = function(){
430         utils.logJSON('logonCore.skipClientHub');
431         logonCore.skipClientHub(onCoreResult,
432 onFlowError);
432     }
433
434     var callPersistWithDefaultPasscode =
435 function(context){
435         utils.logJSON(context,
436 'logonCore.persistRegistration');
436         context.passcode = null;
437         logonCore.persistRegistration(
438             onCoreResult,
439             onFlowError,
440             context)
441     }
442
443     // exported properties
```

```
444         this.stateTransitions = [  
445             {  
446                 condition: {  
447                     state: {  
448                         secureStoreOpen: false,  
449                         status: 'fullRegistered',  
450                         defaultPasscodeUsed: true  
451                     }  
452                 },  
453                 action: onUnlockVaultWithDefaultPasscode  
454             },  
455             {  
456                 condition: {  
457                     state: {  
458                         secureStoreOpen: false,  
459                         status: 'fullRegistered'  
460                     }  
461                 },  
462                 action: 'SCR_UNLOCK'  
463             },  
464             {  
465                 condition: {  
466                     state: {  
467                         //secureStoreOpen: false, //TODO  
468                         clarify  
469                         status: 'fullRegistered',  
470                         stateClientHub: 'availableNoSSOPin'  
471                     }  
472                 },  
473             },  
474         ],
```

```
475         action: 'SCR_SSOPIN_SET'
476     },
477     {
478         condition: {
479             state: {
480                 secureStoreOpen: false,
481                 status: 'new'
482             },
483             context: null
484         },
485         action: callGetContext
486     },
487
488     {
489         condition: {
490             state: {
491                 secureStoreOpen: false,
492                 status: 'new',
493                 stateClientHub: 'availableNoSSOPin'
494             }
495         },
496         action: 'SCR_SSOPIN_SET'
497     },
498
499     {
500         condition: {
501             state: {
502                 secureStoreOpen: false,
503                 status: 'new',
504                 stateClientHub:
505                 'availableInvalidSSOPin'
505             }
506         }
507     }
508 }
```

Kapsel Development

```
506         },
507         action: 'SCR_SSOPIN_SET'
508     },
509
510     {
511         condition: {
512             state: {
513                 secureStoreOpen: false,
514                 status: 'new',
515                 stateClientHub:
516                 'availableValidSSOPin'
517             },
518             context : {
519                 credentialsByClientHub : true,
520                 registrationReadOnly : true
521             }
522         },
523         action: function(context){
524             utils.logJSON(context,
525             'logonCore.startRegistration');
526             logonCore.startRegistration(onCoreResult,
527             onFlowError, context.registrationContext);
528         }
529     },
530
531     {
532         condition: {
533             state: {
534                 secureStoreOpen: false,
535                 status: 'new',
536                 stateClientHub:
537                 'availableValidSSOPin',
538                 stateAfaria:
539                 'initializationSuccessful'
540             },
```



```
535         context : {
536             registrationReadOnly : true,
537             afariaRegistration: 'certificate'
538         }
539     },
540     action: function(context){
541         utils.logJSON(context,
542             'logonCore.startRegistration');
543         logonCore.startRegistration(onCoreResult,
544             onFlowError, context.registrationContext);
545     }
546     },
547     {
548         condition: {
549             state: {
550                 secureStoreOpen: false,
551                 status: 'new',
552                 stateClientHub:
553                 'availableValidSSOPin'
554             },
555             context : {
556                 registrationReadOnly :true,
557                 credentialsByClientHub : false
558             }
559         },
560         action: 'SCR_ENTER_CREDENTIALS'
561     },
562     {
563         condition: {
564             state: {
565                 secureStoreOpen: false,
```

Kapsel Development

```
565                 status: 'new',
566                 //stateClientHub: 'notAvailable' |
'availableValidSSOPin' | 'skipped' | 'error'
567                 stateAfaria: 'initializationFailed'
568             }
569         },
570         action: 'SCR_REGISTRATION'
571     },
572
573     {
574         condition: {
575             state: {
576                 secureStoreOpen: false,
577                 status: 'new',
578                 //stateClientHub: 'notAvailable' |
'availableValidSSOPin' | 'skipped' | 'error'
579             }
580         },
581         action: 'SCR_REGISTRATION'
582     },
583
584     {
585         condition: {
586             state: {
587                 secureStoreOpen: false,
588                 status: 'new',
589                 //stateClientHub: 'notAvailable' |
'availableValidSSOPin' | 'skipped' | 'error'
590             }
591         },
592         action: 'SCR_REGISTRATION'
593     },
594     },
```

```
595
596     {
597         condition: {
598             state: {
599                 secureStoreOpen: false,
600                 status: 'registered',
601                 defaultPasscodeUsed: true,
602                 // defaultPasscodeAllowed: true,
603             }
604         },
605         action: 'SCR_SET_PASSCODE_OPT_OFF'
606     },
607     {
608         condition: {
609             state: {
610                 secureStoreOpen: false,
611                 status: 'registered',
612                 defaultPasscodeUsed: false,
613                 defaultPasscodeAllowed: true,
614             }
615         },
616         action: 'SCR_SET_PASSCODE_OPT_ON'
617     },
618     {
619         condition: {
620             state: {
621                 secureStoreOpen: false,
622                 status: 'registered',
623                 // defaultPasscodeAllowed: false,
624             }
625         },
626         action: 'SCR_SET_PASSCODE_MANDATORY'
```

Kapsel Development

```
627         },
628
629
630     {
631         condition: {
632             state: {
633                 //secureStoreOpen: false, //TODO
clarify
634                 status: 'fullRegistered',
635                 stateClientHub:
'availableInvalidSSOPin'
636             }
637         },
638         action: 'SCR_SSOPIN_CHANGE'
639     },
640     {
641         condition: {
642             state: {
643                 secureStoreOpen: true,
644                 status: 'fullRegistered',
645                 stateClientHub: 'notAvailable'
646             }
647         },
648         action: onFullRegistered
649     },
650     {
651         condition: {
652             state: {
653                 secureStoreOpen: true,
654                 status: 'fullRegistered',
655                 stateClientHub:
'availableValidSSOPin'
656             }
```

```
657         },
658         action: onFullRegistered
659     },
660     {
661         condition: {
662             state: {
663                 secureStoreOpen: true,
664                 status: 'fullRegistered',
665                 stateClientHub: 'skipped'
666             }
667         },
668         action: onFullRegistered
669     },
670
671
672
673     ];
674
675     this.screenEvents = {
676         'SCR_SSOPIN_SET': {
677             onsubmit: function(context){
678                 utils.logJSON(context,
679                 'logonCore.setSSOPasscode');
680                 logonCore.setSSOPasscode(onCoreResult,
681                 onSSOPasscodeSetError, context);
682             },
683             oncancel: onCancelSSOPin,
684             onerror: onFlowError,
685             onforgot: onForgotSsoPin,
686             onskip: onSkipSsoPin
687         },
688         'SCR_SSOPIN_CHANGE': {
```

```
688             onSubmit: function(context){
689                 utils.logJSON(context,
'loginCore.setSSOPasscode');
690                 loginCore.setSSOPasscode(onCoreResult,
onSSOPasscodeSetError, context);
691             },
692             onCancel: onSkipSsoPin,
693             onError: onFlowError,
694             onforgot: onForgotSsoPin
695         },
696
697         'SCR_UNLOCK': {
698             onSubmit: onUnlockSubmit,
699             onCancel: noOp,
700             onError: onFlowError,
701             onforgot: onForgotAppPasscode,
702             onerrorack: noOp
703         },
704
705         'SCR_REGISTRATION': {
706             onSubmit: onRegSubmit,
707             onCancel: onCancelRegistration,
708             onError: onFlowError,
709             onbackbutton: onRegistrationBackButton
710         },
711
712         'SCR_ENTER_CREDENTIALS' : {
713             onSubmit: onRegSubmit,
714             onCancel: onCancelRegistration,
715             onError: onFlowError
716         },
717         'SCR_SET_PASSCODE_OPT_ON': {
718             onSubmit: onCreatePasscodeSubmit,
```

```
719             onCancel: noOp,  
720             onError: onFlowError,  
721             onDisable:  
showScreen('SCR_SET_PASSCODE_OPT_OFF'),  
722             onErrorack: noOp  
723         },  
724         'SCR_SET_PASSCODE_OPT_OFF': {  
725             onsubmit:  
callPersistWithDefaultPasscode,  
726             onCancel: noOp,  
727             onError: onFlowError,  
728             onEnable:  
showScreen('SCR_SET_PASSCODE_OPT_ON'),  
729             onErrorack: noOp  
730         },  
731         'SCR_SET_PASSCODE_MANDATORY': {  
732             onsubmit: onCreatePasscodeSubmit,  
733             onCancel: noOp,  
734             onError: onFlowError,  
735             onErrorack: noOp  
736         },  
737  
738  
739  
740     };  
741  
742  
743     utils.log('flow constructor return');  
744 }  
745  
746  
747
```

Kapsel Development

```
748         var ChangePasswordFlow = function
ChangePasswordFlow(logonCore, logonView, onCoreResult,
onFlowSuccess, onFlowError, onFlowCancel) {
749             //wrapped into a function to defer evaluation of the
references to flow callbacks
750
751             this.name = 'changePasswordFlowBuilder';
752
753
754             // internal methods
755
756             var callUnlockFlow = function(){
757                 utils.log(this.name + ' triggered
unlock');
758                 registerOrUnlock(onCoreResult, onFlowError);
759             }
760
761             var onChangePasswordSubmit = function(context){
762                 utils.logJSON(context,
'logonCore.changePassword');
763                 // this logonCore call does not return with
context
764                 logonCore.changePassword(onPasswordChanged,
onFlowError, context);
765             }
766
767
768             var onPasswordChanged = function(){
769                 utils.log('onPasswordChanged');
770                 logonCore.getContext(onFlowSuccess,
onFlowError);
771             }
772
773             // exported properties
774             this.stateTransitions = [
```



```
775         {
776             condition: {
777                 state: {
778                     secureStoreOpen: false,
779                 }
780             },
781             action: callUnlockFlow,
782         },
783     {
784         condition: {
785             state: {
786                 secureStoreOpen: true,
787             }
788         },
789         action: 'SCR_CHANGE_PASSWORD'
790     },
791 ];
792
793
794     this.screenEvents = {
795         'SCR_CHANGE_PASSWORD': {
796             onsubmit: onChangePasswordSubmit,
797             oncancel: onFlowCancel,
798             onerror: onFlowError
799         }
800     };
801
802
803     utils.log('flow constructor return');
804 }
805
```

Kapsel Development

```
806         var ManagePasscodeFlow = function
ManagePasscodeFlow(logonCore, logonView, onCoreResult,
onFlowSuccess, onFlowError, onFlowCancel) {
807             //wrapped into a function to defer evaluation of the
references to flow callbacks
808
809             this.name = 'managePasscodeFlowBuilder';
810
811             // internal methods
812             var showScreen = function(screenId) {
813                 return function(coreContext) {
814                     logonView.showScreen(screenId,
this.screenEvents[screenId], coreContext);
815                 }.bind(this);
816             }.bind(this);
817
818
819             var callChangePasscode = function(context){
820                 utils.logJSON(context,
'logonCore.changePasscode');
821                 logonCore.changePasscode(
822                     onCoreResult,
823                     onChangePasscodeError,
824                     context)
825             }
826
827             var onChangePasscodeError = function(error) {
828                 utils.logJSON("onChangePasscodeError: " +
JSON.stringify(error));
829                 logonView.showNotification(getSecureStoreErrorText(error));
830             }
831
832             var noOp = function() { }
833
```

```
834         var callDisablePasscode = function(context){
835             utils.logJSON(context,
'logonCore.disablePasscode');
836             context.passcode = null;
837             logonCore.changePasscode(
838                 onCoreResult,
839                 onFlowError,
840                 context)
841         }
842
843         var callGetContext = function(){
844             utils.log('logonCore.getContext');
845             logonCore.getContext(onCoreResult,
onFlowError);
846         }
847
848         var onPasscodeEnable = function(context){
849             utils.logJSON(context, this.name + '
onPasscodeEnable: ');
850             //logonCore.changePasscode(onFlowSuccess,
onFlowError, context);
851             onFlowError();
852         }
853
854         // exported properties
855         this.stateTransitions = [
856             {
857                 condition: {
858                     state: {
859                         secureStoreOpen: true,
860                     },
861                 context: null
862             },
863             action: callGetContext
```

```
864         },
865         {
866             condition: {
867                 state: {
868                     secureStoreOpen: false,
869                 }
870             },
871             action: onFlowError
872         },
873         {
874             condition: {
875                 state: {
876                     secureStoreOpen: true,
877                     defaultPasscodeUsed: true,
878                     // defaultPasscodeAllowed: true,
879                 }
880             },
881             action: 'SCR_MANAGE_PASSCODE_OPT_OFF'
882         },
883         {
884             condition: {
885                 state: {
886                     secureStoreOpen: true,
887                     defaultPasscodeUsed: false,
888                     defaultPasscodeAllowed: true,
889                 }
890             },
891             action: 'SCR_MANAGE_PASSCODE_OPT_ON'
892         },
893         {
894             condition: {
895                 state: {
```

```
896         secureStoreOpen: true,
897         //defaultPasscodeUsed: [DONTCARE],
898         defaultPasscodeAllowed: false,
899     }
900 },
901     action: 'SCR_MANAGE_PASSCODE_MANDATORY'
902 },
903
904
905 ];
906
907     this.screenEvents = {
908         'SCR_MANAGE_PASSCODE_OPT_ON': {
909             onsubmit: onFlowSuccess,
910             oncancel: onFlowSuccess,
911             onerror: onFlowError,
912             ondisable:
913 showScreen('SCR_CHANGE_PASSCODE_OPT_OFF'),
914             onchange:
915 showScreen('SCR_CHANGE_PASSCODE_OPT_ON')
916         },
917         'SCR_MANAGE_PASSCODE_OPT_OFF': {
918             onsubmit: onFlowSuccess,
919             oncancel: onFlowSuccess,
920             onerror: onFlowError,
921             onenable:
922 showScreen('SCR_SET_PASSCODE_OPT_ON')
923         },
924         'SCR_MANAGE_PASSCODE_MANDATORY': {
925             onsubmit: onFlowSuccess,
926             oncancel: onFlowSuccess,
927             onerror: onFlowError,
928             onchange:
929 showScreen('SCR_CHANGE_PASSCODE_MANDATORY')
```

```
926         },
927
928
929         'SCR_SET_PASSCODE_OPT_ON': {
930             onsubmit: callChangePasscode,
931             oncancel: onFlowCancel,
932             onerror: onFlowError,
933             ondisable:
showScreen('SCR_SET_PASSCODE_OPT_OFF'),
934             onerrorack: noOp
935         },
936         'SCR_SET_PASSCODE_OPT_OFF': {
937             onsubmit: callDisablePasscode,
938             oncancel: onFlowCancel,
939             onerror: onFlowError,
940             onenable:
showScreen('SCR_SET_PASSCODE_OPT_ON'),
941             onerrorack: noOp
942         },
943         'SCR_CHANGE_PASSCODE_OPT_ON': {
944             onsubmit: callChangePasscode,
945             oncancel: onFlowCancel,
946             onerror: onFlowError,
947             ondisable:
showScreen('SCR_CHANGE_PASSCODE_OPT_OFF'),
948             onerrorack: noOp
949         },
950         'SCR_CHANGE_PASSCODE_OPT_OFF': {
951             onsubmit: callDisablePasscode,
952             oncancel: onFlowCancel,
953             onerror: onFlowError,
954             onenable:
showScreen('SCR_CHANGE_PASSCODE_OPT_ON'),
955             onerrorack: noOp
```

```
956         },
957         'SCR_CHANGE_PASSCODE_MANDATORY': {
958             onsubmit: callChangePasscode,
959             oncancel: onFlowCancel,
960             onerror: onFlowError,
961             onerrorack: noOp
962         },
963
964     };
965
966
967     utils.log('flow constructor return');
968 }
969
970     var ShowRegistrationFlow = function
ShowRegistrationFlow(logonCore, logonView, onCoreResult,
onFlowSuccess, onFlowError, onFlowCancel) {
971     //wrapped into a function to defer evaluation of the
references to flow callbacks
972
973     this.name = 'showRegistrationFlowBuilder';
974
975     var showRegistrationInfo = function(context) {
976         logonView.showScreen('SCR_SHOW_REGISTRATION',
this.screenEvents['SCR_SHOW_REGISTRATION'], context);
977     }.bind(this);
978
979     var callGetContext = function(){
980         utils.log('logonCore.getContext');
981         logonCore.getContext(onCoreResult,
onFlowError);
982     }
983
984     // exported properties
```

```
985         this.stateTransitions = [  
986             {  
987                 condition: {  
988                     state: {  
989                         secureStoreOpen: true,  
990                     },  
991                 },  
992                 context: null  
993             },  
994             action: callGetContext  
995         },  
996         {  
997             condition: {  
998                 secureStoreOpen: true,  
999             },  
1000             action: showRegistrationInfo  
1001         }  
1002     ];  
1003  
1004  
1005     this.screenEvents = {  
1006         'SCR_SHOW_REGISTRATION': {  
1007             oncancel: onFlowSuccess,  
1008             onerror: onFlowError  
1009         }  
1010     };  
1011  
1012  
1013     utils.log('flow constructor return');  
1014 }  
1015
```



```
1016 // === flow launcher methods
=====
1017
1018
1019     var resume = function (onsuccess, onerror) {
1020
1021         var onUnlockSuccess = function(){
1022             _oLogonCore.onEvent(onsuccess, onerror,
1023 'RESUME');
1024         }
1025
1026         var onGetStateSuccess = function(state) {
1027             //call registration flow only if the status is
1028             fullregistered in case of resume, so logon screen will not loose its
1029             input values
1030             if (state.status == 'fullRegistered') {
1031                 registerOrUnlock(onUnlockSuccess,
1032 onerror);
1033             }
1034         }
1035
1036         var get = function (onsuccess, onerror, key) {
1037
1038             var onUnlockSuccess = function(){
1039                 _oLogonCore.getSecureStoreObject(onsuccess,
1040 onerror, key);
1041             }
1042
1043             registerOrUnlock(onUnlockSuccess, onerror);
1044         }
1045     }
```

Kapsel Development

```
1044
1045
1046
1047     var set = function (onsuccess, onerror, key, value)
1048     {
1049         var onUnlockSuccess = function(){
1050             _oLogonCore.setSecureStoreObject(onsuccess,
1051             onerror, key, value);
1052         }
1053         registerOrUnlock(onUnlockSuccess, onerror);
1054     }
1055
1056
1057
1058     var lock = function (onsuccess, onerror) {
1059         _oLogonCore.lockSecureStore(onsuccess,
1060         onerror);
1061     }
1062
1063     var getState = function (onsuccess, onerror) {
1064         _oLogonCore.getState(onsuccess, onerror);
1065     }
1066
1067     var registerOrUnlock = function(onsuccess, onerror)
1068     {
1069         var flowRunner = new FlowRunner(onsuccess, onerror,
1070         _oLogonView, _oLogonCore);
1071         flowRunner.run(RegistrationFlow);
1072     }
1073
1074     var changePassword = function(onsuccess, onerror) {
```

```
1073
1074         var onUnlockSuccess = function(){
1075             var innerFlowRunner = new FlowRunner(onsuccess,
1076             onerror, _oLogonView, _oLogonCore);
1077             innerFlowRunner.run(ChangePasswordFlow);
1078         }
1079         registerOrUnlock(onUnlockSuccess, onerror);
1080     }
1081
1082
1083     var forgottenPasscode = function(onsuccess, onerror)
1084     {
1085         var onUnlockSuccess = function(){
1086             var innerFlowRunner = new FlowRunner(onsuccess,
1087             onerror, _oLogonView, _oLogonCore);
1088             innerFlowRunner.run(MockFlow);
1089         }
1090         registerOrUnlock(onUnlockSuccess, onerror);
1091     }
1092
1093     var managePasscode = function(onsuccess, onerror) {
1094
1095         var onUnlockSuccess = function(){
1096             var innerFlowRunner = new FlowRunner(onsuccess,
1097             onerror, _oLogonView, _oLogonCore);
1098             innerFlowRunner.run(ManagePasscodeFlow);
1099         }
1100         registerOrUnlock(onUnlockSuccess, onerror);
1101     }
1102
```

```
1103         var showRegistrationData = function(onsuccess,
onerror) {
1104             var onUnlockSuccess = function(){
1105                 var innerFlowRunner = new FlowRunner(onsuccess,
onerror, _oLogonView, _oLogonCore);
1106                 innerFlowRunner.run(ShowRegistrationFlow);
1107             }
1108
1109             registerOrUnlock(onUnlockSuccess, onerror);
1110         }
1111
1112         var getSecureStoreErrorText = function(error) {
1113             utils.logJSON('LogonController.getSecureStoreErrorText: ' +
JSON.stringify(error));
1114
1115             var errorText;
1116
1117             if(error.errorCode === '14' && error.errorDomain
=== 'MAFSecureStoreManagerErrorDomain')
1118                 errorText = "ERR_PASSCODE_TOO_SHORT";
1119             else if(error.errorCode === '10' &&
error.errorDomain === 'MAFSecureStoreManagerErrorDomain')
1120                 errorText = "ERR_PASSCODE_REQUIRES_DIGIT";
1121             else if(error.errorCode === '13' &&
error.errorDomain === 'MAFSecureStoreManagerErrorDomain')
1122                 errorText = "ERR_PASSCODE_REQUIRES_UPPER";
1123             else if(error.errorCode === '11' &&
error.errorDomain === 'MAFSecureStoreManagerErrorDomain')
1124                 errorText = "ERR_PASSCODE_REQUIRES_LOWER";
1125             else if(error.errorCode === '12' &&
error.errorDomain === 'MAFSecureStoreManagerErrorDomain')
1126                 errorText =
"ERR_PASSCODE_REQUIRES_SPECIAL";
1127             else if(error.errorCode === '15' &&
error.errorDomain === 'MAFSecureStoreManagerErrorDomain')
```

```
1128         errorText =
1129         "ERR_PASSCODE_UNDER_MIN_UNIQUE_CHARS";
1130     else {
1131         errorText = "ERR_SETPASSCODE_FAILED";
1132     }
1133     return errorText;
1134 }
1135
1136     var getSSOPasscodeSetErrorText = function(error) {
1137     utils.logJSON('LogonController.getSSOPasscodeSetErrorText: ' +
1138     JSON.stringify(error));
1139     var errorText;
1140
1141     if (error.errorDomain ===
1142     'MAFLogonCoreErrorDomain') {
1143         if (error.errorCode === '16') {
1144             errorText =
1145             "ERR_SSO_PASSCODE_SET_ERROR";
1146         }
1147     }
1148     return errorText;
1149 }
1150     var getRegistrationErrorText = function(error) {
1151     utils.logJSON('LogonController.getRegistrationErrorText: ' +
1152     JSON.stringify(error));
1153     var errorText;
1154 }
```

```
1155         if (error.errorDomain ===
'MAFLogonCoreErrorDomain') {
1156             if (error.errorCode === '80003') {
1157                 errorText =
"ERR_REG_FAILED_WRONG_SERVER";
1158             }
1159             //in case of wrong application id
1160             else if (error.errorCode === '404') {
1161                 errorText = "ERR_REG_FAILED";
1162             }
1163             else if (error.errorCode === '401') {
1164                 errorText =
"ERR_REG_FAILED_UNAUTHORIZED";
1165             }
1166             else {
1167                 errorText = "ERR_REG_FAILED";
1168             }
1169         }
1170
1171         return errorText;
1172     }
1173
1174     var getRegistrationCancelError = function(error) {
1175         utils.logJSON('LogonController.getRegistrationCancelError: ' +
JSON.stringify(error));
1176
1177         var errorText;
1178
1179         errorText = "ERR_REGISTRATION_CANCEL";
1180
1181         return errorText;
1182     }
1183
```

```

1184         var errorWithDomainCodeDescription = function(domain,
code, description) {
1185             var error = {
1186                 errorDomain: domain,
1187                 errorCode: code,
1188                 errorMessage: description
1189             };
1190
1191             return error;
1192         }
1193
1194
1195
1196
1197     // ===== exported (public) members
=====
1198
1199     /**
1200     * The Logon plugin provides screen flows to register an
app with an SAP Mobile Platform server.<br/>
1201     * <br/>
1202     * The logon plugin is a component of the SAP Mobile
Application Framework (MAF), exposed as a Cordova plugin. The basic
1203     * idea is that it provides screen flows where the user can
enter the values needed to connect to an SAP Mobile Platform 3.0
server and
1204     * stores those values in its own secure data vault. This
data vault is separate from the one provided with the
1205     * encrypted storage plugin. In an OData based SAP Mobile
Platform 3.0 application, a client must onboard or register with the
SAP Mobile Platform 3.0
1206     * server to receive an application connection ID for a
particular app. The application connection ID must be sent
1207     * along with each request that is proxied through the SAP
Mobile Platform 3.0 server to the OData producer.<br/>
1208     * <br/>
1209     * <b>Adding and Removing the Logon Plugin</b><br/>

```

Kapsel Development

```
1210      * The Logon plugin is added and removed using the
1211      * <a href="http://cordova.apache.org/docs/en/edge/
guide_cli_index.md.html#The%20Command-line%20Interface">Cordova
CLI</a>.<br/>
1212      * <br/>
1213      * To add the Logon plugin to your project, use the
following command:<br/>
1214      * cordova plugin add <full path to directory containing
Kapsel plugins>\logon<br/>
1215      * <br/>
1216      * To remove the Logon plugin from your project, use the
following command:<br/>
1217      * cordova plugin rm com.sap.mp.cordova.plugins.logon
1218      *
1219      * @namespace
1220      * @alias Logon
1221      * @memberof sap
1222      */
1223      module.exports = {
1224
1225      /**
1226      * Initialization method to set up the Logon plugin. This
will register the application with the SMP server and also
authenticate the user
1227      * with servers on the network. This step must be done
first prior to any attempt to communicate with the SMP server.
1228      *
1229      * @method
1230      * @param {sap.Logon~successCallback} successCallback The
function that is invoked if initialization is successful. The
current
1231      * context is passed to this function as the parameter.
1232      * @param {sap.Logon~errorCallback} errorCallback The
function that is invoked in case of an error.
1233      * @param {string} applicationId The unique ID of the
application. Must match the application ID on the SAP Mobile
Platform server.
```



```

1234      * @param {object} [context] The context with default
values for application registration. See {@link
sap.Logon~successCallback} for the structure

1235      * of the context object. Note that all properties of the
context object are optional, and you only need to specify the
properties

1236      * for which you want to provide default values for. The
values will be presented to the application users during the
registration process and given them

1237      * a chance to override these values during runtime.

1238      * @param {string} [logonView="com/sap/mp/logon/iabui"]
The cordova module ID of a custom renderer for the logon,

1239      * implementing the [showScreen(), close()] interface.
Please use the default module unless you are absolutely sure that you
can provide your own

1240      * custom implementation. Please refer to JavaScript
files inside your Kapsel project's plugins\logon\www\common\modules\
folder as example.

1241      * @example

1242      * // a custom UI can be loaded here

1243      * var logonView = sap.logon.IabUi;

1244      *

1245      * // The app ID

1246      * var applicationId = "someAppID";

1247      *

1248      * // You only need to specify the fields for which you
want to set the default. These values are optional because they
will be

1249      * // used to prefill the fields on Logon's UI
screen.

1250      * var defaultContext = {

1251      *   "serverHost" : "defaultServerHost.com"

1252      *   "https" : false,

1253      *   "serverPort" : "8080",

1254      *   "user" : "user1",

1255      *   "password" : "Zzzzzz123",

1256      *   "communicatorId" : "REST",

1257      *   "securityConfig" : "sec1",

```

Kapsel Development

```
1258     *     "passcode" : "Aaaaaa123",
1259     *     "unlockPasscode" : "Aaaaaa123"
1260     * };
1261     *
1262     * var app_context;
1263     *
1264     * var successCallback = function(context) {
1265     *     app_context = context;
1266     * }
1267     *
1268     * var errorCallback = function(errorInfo) {
1269     *     alert("error: " + JSON.stringify(errorInfo));
1270     * }
1271     * sap.Logon.init(successCallback, errorCallback,
1272     * applicationId, defaultContext, logonView);
1273     * /
1274     *
1275     * /**
1276     * The application ID with which {@link sap.Logon.init}
1277     * was called. It is available here so it is easy to access later.
1278     * @example
1279     * // After calling the init function
1280     * alert("The app ID for this app is: " +
1281     * sap.Logon.applicationId);
1282     * /
1283     *
1284     * applicationId: null,
1285     * /**
1286     * Direct reference to the logon core object used by the
1287     * Logon plugin. This is needed to perform more complex operations
1288     * that
1289     * are not generally needed by applications. <br/>
1290     * There are several functions that can be accessed on the
1291     * core object:<br/>
```


Kapsel Development

```
1308      * This can be null or undefined, if no object is defined
for the given key.
1309      * @param {sap.Logon~errorCallback} onerror The function to
invoke in case of error.
1310      * @param {string} key The key with which to query the
DataVault.
1311      * @example
1312      * var errorCallback = function(errorInfo){
1313      *     alert("Error: " + JSON.stringify(errorInfo));
1314      * }
1315      * var getSuccess = function(value){
1316      *     alert("value retrieved from the store: " +
JSON.stringify(value));
1317      * }
1318      * var setSuccess = function(){
1319      *
sap.Logon.get(getSuccess,errorCallback,'someKey');
1320      * }
1321      * sap.Logon.set(setSuccess,errorCallback,'someKey',
'some string (could also be an object).');
1322      */
1323      get: get,
1324
1325      /**
1326      * Set an (JSON serializable) object in the DataVault.
1327      * @method
1328      * @param {sap.Logon~successCallbackNoParameters}
onsuccess The function to invoke upon success.
1329      * onsuccess will be called without parameters for this
method.
1330      * @param {sap.Logon~errorCallback} onerror The function to
invoke in case of error.
1331      * @param {string} key The key to store the provided
object on.
1332      * @param {object} value The object to be set on the given
key. Must be JSON serializable (ie:
1333      * cannot contain circular references).
```



```
1363
1364     /**
1365         * Unlock the Logon plugin's secure data vault if it has
1366         * been locked (due to being inactive, or
1367         * {@link sap.Logon.lock} being called), then the user is
1368         * prompted for the passcode to unlock the
1369         * application.<br/>
1370         * If the application is already unlocked, then nothing
1371         * will be done.<br/>
1372         * If the application has passcode disabled, then passcode
1373         * prompt will not be necessary.
1374         * In all cases if an error does not occur, the success
1375         * callback is invoked with the current logon context
1376         * as the parameter.
1377         * @method
1378         * @param {sap.Logon~successCallback} onSuccess - The
1379         * callback to call if the screen flow succeeds.
1380         * onSuccess will be called with the current logon context
1381         * as a single parameter.
1382         * @param {sap.Logon~errorCallback} onError - The callback
1383         * to call if the screen flow fails.
1384         * @example
1385         * var errorCallback = function(errorInfo){
1386         *     alert("Error: " + JSON.stringify(errorInfo));
1387         * }
1388         * var successCallback = function(context){
1389         *     alert("Registered and unlocked. Context: " +
1390         *     JSON.stringify(context));
1391         * }
1392         * sap.Logon.unlock(successCallback,errorCallback);
1393     */
1394     unlock: registerOrUnlock,
1395
1396     /**
1397         * This is an alias for registerOrUnlock. Calling this
1398         * function is equivalent
```

```

1389      * to calling {@link sap.Logon.unlock} since both of them
are alias to registerOrUnlock.
1390      * @method
1391      * @private
1392      */
1393      registerUser: registerOrUnlock,
1394
1395      /**
1396      * This function registers the user and creates a new
unlocked DataVault to store the registration
1397      * information.<br/>
1398      * If the user has already been registered, but the
application is locked (due to being inactive, or
1399      * {@link sap.Logon.lock} being called), then the user is
prompted for the passcode to unlock the
1400      * application.<br/>
1401      * If the application is already unlocked, then nothing
will be done.<br/>
1402      * In all cases if an error does not occur, the success
callback is invoked with the current logon context
1403      * as the parameter.
1404      * @method
1405      * @param {sap.Logon~successCallback} onSuccess - The
callback to call if the screen flow succeeds.
1406      * onSuccess will be called with the current logon context
as a single parameter.
1407      * @param {sap.Logon~errorCallback} onError - The callback
to call if the screen flow fails.
1408      * @example
1409      * var errorCallback = function(errorInfo){
1410      *     alert("Error: " + JSON.stringify(errorInfo));
1411      * }
1412      * var successCallback = function(context){
1413      *     alert("Registered and unlocked. Context: " +
JSON.stringify(context));
1414      * }

```

Kapsel Development

```
1415      *
sap.Logon.registerOrUnlock(successCallback,errorCallback);

1416      * @private

1417      */

1418      registerOrUnlock: registerOrUnlock,

1419

1420      /**

1421      * This method will launch the UI screen for application
users to manage and update the data vault passcode or,

1422      * if the SMP server's Client Passcode Policy allows it,
enable or disable the passcode to the data vault.

1423      *

1424      * @method

1425      * @param {sap.Logon~successCallbackNoParameters}
onsuccess - The function to invoke upon success.

1426      * @param {sap.Logon~errorCallback} onerror - The function
to invoke in case of error.

1427      * @example

1428      * var errorCallback = function(errorInfo){

1429      *     alert("Error: " + JSON.stringify(errorInfo));

1430      * }

1431      * var successCallback = function(context){

1432      *     alert("Passcode successfully managed.");

1433      * }

1434      *
sap.Logon.managePasscode(successCallback,errorCallback);

1435      */

1436      managePasscode: managePasscode,

1437

1438      /**

1439      * This method will launch the UI screen for application
users to manage and update the back end passcode that Logon stores in
the

1440      * data vault that is used to authenticate the client to
the server.

1441      *
```



```

1442      * @method
1443      * @param {sap.Logon~successCallbackNoParameters}
onsuccess - The callback to call if the screen flow succeeds.
1444      * onsuccess will be called without parameters for this
method.
1445      * @param {sap.Logon~errorCallback} onerror The function
that is invoked in case of an error.
1446      * @example
1447      * var errorCallback = function(errorInfo){
1448      *     alert("Error: " + JSON.stringify(errorInfo));
1449      * }
1450      * var successCallback = function(context){
1451      *     alert("Password successfully changed.");
1452      * }
1453      *
sap.Logon.changePassword(successCallback,errorCallback);
1454      */
1455      changePassword: changePassword,
1456
1457      /**
1458      * Calling this method will show a UI screen with values
used for registrating application.
1459      * @method
1460      * @param {sap.Logon~successCallbackNoParameters}
onsuccess - The callback to call if the screen flow succeeds.
1461      * onsuccess will be called without parameters for this
method.
1462      * @param {sap.Logon~errorCallback} onerror The function
that is invoked in case of an error.
1463      * @example
1464      * var errorCallback = function(errorInfo){
1465      *     alert("Error: " + JSON.stringify(errorInfo));
1466      * }
1467      * var successCallback = function(context){
1468      *     alert("The showRegistrationData screenflow was
successful.");

```

Kapsel Development

```
1469         * }
1470         *
sap.Logon.showRegistrationData(successCallback,errorCallback);
1471         */
1472         showRegistrationData: showRegistrationData,
1473
1474     };
1475
1476     /**
1477     * Callback function that is invoked in case of an error.
1478     *
1479     * @callback sap.Logon~errorCallback
1480     *
1481     * @param {Object} errorObject An object containing
properties: 'errorCode', 'errorMessage', and 'errorDomain'.
1482     * The 'errorCode' is just a number uniquely identifying the
error. The 'errorMessage'
1483     * property is a string with more detailed information of what
went wrong. The 'errorDomain' property specifies
1484     * the domain that the error occurred in.
1485     */
1486
1487     /**
1488     * Callback function that is invoked upon successfully
registering or unlocking or retrieving the context.
1489     *
1490     * @callback sap.Logon~successCallback
1491     *
1492     * @param {Object} context An object containing the current
logon context. Two properties of particular importance
1493     * are applicationEndpointURL, and applicationConnectionId.
1494     * The context object contains the following properties:<br/>
1495     * "registrationContext": {<br/>
```


AuthProxy Plugin Overview

The AuthProxy plugin provides the ability to make HTTPS requests with mutual authentication.

The AuthProxy plugin allows you to specify a certificate to include in an HTTPS request that identifies the client to the server, which allows the server to verify the identity of the client. An example of where you might need mutual authentication is in the onboarding process, when you register with an application, or, to access an OData producer. You can make HTTPS requests with no authentication, with basic authentication, or by using certificates. Supported certificate sources include file, system key manager, and Afaria.

Sending Requests

There are two functions for sending requests:

- `get = function (url, header, successCB, errorCallback, user, password, timeout, certSource).`
This is a convenience function and provides no additional functionality compared to the `sendRequest` function. It just calls the `sendRequest` function with the method set to `GET` and no `requestBody`.
- `sendRequest = function (method, url, header, requestBody, successCB, errorCallback, user, password, timeout, certSource).`

Constructor Functions

There are three constructor functions to make objects that you can use for certificates:

- `CertificateFromFile = function (Path, Password, CertificateKey)`
- `CertificateFromLogonManager = function(AppID)`
- `CertificateFromStore = function (CertificateKey)`

Note: The `success` callback is called upon any response from the server, so be sure to check the status on the response.

Adding the AuthProxy Plugin

Use the Cordova command line interface to install the AuthProxy plugin.

Prerequisites

- Set up the development environment.
- Create your Cordova Project.
- Add your OS platforms.
- On Android these permissions are required:
 - `<uses-permission android:name="android.permission.INTERNET" />`
 - `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />`

- `<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />`
- On iOS:
 - The plugin depends on `afariaSSL.a`
 - Requires the link flag of `"-lstdc++,"` if not yet included.

Task

1. Add the AuthProxy plugin by entering the following at the command prompt, or terminal:

On Windows:

```
cordova -d plugin add <SDK_HOME>\MobileSDK3\KapselSDK
\plugins\authproxy
```

On Mac:

```
cordova -d plugin add ~<SDK_HOME>/MobileSDK3/KapselSDK/
plugins/authproxy
```

Note: The path you enter to the Kapsel plugin must be the absolute path (not relative path).

2. (Optional) To see a list of installed plugins in your Cordova project, open a command prompt or terminal window, navigate to your Cordova project folder, and enter:

```
cordova plugins
```

The Cordova command line interface returns a JSON array showing installed plugins, for example:

```
[ 'org.apache.cordova.core.camera',
  'org.apache.cordova.core.device-motion',
  'org.apache.cordova.core.file' ]
```

In this example, the Cordova project has the Cordova core Camera, Accelerator (device-motion), and File plugins installed.

3. Modify the files in the `www` folder for the project as necessary, then copy them to the platform directories by running:

```
cordova -d prepare android
cordova -d prepare ios
```

4. Use the Android IDE or Xcode to deploy and run the project.
5. (Optional) For iOS, if the application uses Afaria mutual certificate authentication, or if multiple applications on the devices need to share the credentials, you must first build and deploy Client Hub to the device, and then add the `"clienthubEntitlements"` and `"$(CFBundleIdentifier)"` items to the shared keychain groups in the application's project settings in Xcode.

Adding User Permissions to the Android Manifest File

Add user permissions to the Android project.

1. In the Android IDE, open the AndroidManifest.xml file.
2. Add user permissions in the AndroidManifest.xml file, for example:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/
android"
    package="smp.tutorial.android"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />
    <uses-permission android:name="android.permission.INTERNET">
    <uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE">
    <uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE">
    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
    </application>
</manifest>
```

3. Select **File > Save**.

Adding Cookies to a Request

To add cookies to a request for authentication, use the header object that is passed to the `get`/`sendRequest` functions.

Only the cookie name and value should be set this way. The other pieces of the cookie (domain, path, and so on) are set automatically based on the URL the request is made against. The cookie is treated as a session cookie and sent on future requests as appropriate. The API examples below show an example of how to set a cookie with the header object.

```
var successCallback = function( result ){
    if( result.status === 200 ) {
        alert("success\!
        Response text: " + result.responseText );
    } else {
        alert("Not success, response status:
        " + result.status);
    }
}
```



```

var failureCallback = function( error ) {
    alert("Error! Code: " + error.errorCode + "\n" +
error.description + "\nNative error code: " +
error.nativeErrorCode );
}

// setting a cookie with a request
var header = {cookie:
"customCookieName=customCookieValue;anotherName=AnotherValue"};

sap.AuthProxy.sendRequest("POST", "http://www.example.com/stuff/
etc", header, null, successCallback, failureCallback);

```

Using the AuthProxy Plugin to Register With SAP Mobile Platform Server

This example procedure demonstrates how to use the AuthProxy plugin to register with the SAP Mobile Platform Server using a client certificate.

This example does not use the Logon plugin to perform the registration. You can test certificates on an Android device or emulator, or an iOS device. The server certificate must be installed on the device's system store, so for iOS, the actual device is required.

1. Use the keytool utility to create the server and client certificates.
The SAP Mobile Platform Server stores its certificates in a file named `smp_keystore.jks`.
2. Download the certificate and generate a certificate signing request (CSR).
3. Import the signed certificate into the keystore.
4. Copy the client's public key to `smp_keystore.jks` so that the server can authenticate the client.
5. Create a security profile in Management Cockpit
6. Import the public and private key of the client certificate to the mobile device using the PKCS12 format.

Both the client certificate (stored in the keystore `client.p12` containing the public and private keys) and the certificate authority's certificate, must be added to the mobile device. You should add the certificate authority's certificate to the device's trust store. The client certificate in this example for Android is placed in a location the application can access it from.

```

adb push SAPServerCA.cer /mnt/sdcard/
adb push client.p12 /mnt/sdcard/
adb shell
cd /mnt/sdcard
ls
exit

```

For an iOS device, both certificates can be installed into the device's trusted store by sending them through an e-mail, opening the device browser to a Web page that contains the links to the certificates, or by using the iPhone Configuration Utility. See <http://support.apple.com/kb/DL1465>.

On the iOS device, the certificates can be viewed and uninstalled under **Settings > General > Profiles**.

In addition to accessing the certificate from the file system and the device's secure store, the client certificate can be provisioned to the device using Afaia and then accessed from Afaia using the Logon plugin using the method

```
sap.AuthProxy.CertificateFromLogonManager("clientKey").
```

7. Create a new Cordova project to perform mutual authentication to the SAP Mobile Platform Server.
8. Add the AuthProxy plugin.
9. Create a new security provider and add an x.509 User Certificate authentication provider.
10. Copy the files to the platform directory by running the **prepare** command.
11. Use the Android IDE or Xcode to deploy and run the project.

Generating Certificates and Keys

Use a PKI system and a trusted CA to generate production-ready certificates and keys that encrypt communication among different SAP Mobile Platform components. You can then use the **keytool** utility to import and export certificate to the keystore.

Note: Any changes to the keystore require the server to be restarted.

Kapsel AuthProxy API Reference

The Kapsel AuthProxy API Reference provides usage information for AuthProxy API classes and methods, as well as provides sample source code.

AuthProxy namespace

The AuthProxy plugin provides the ability to make HTTPS requests with mutual authentication.

The regular XMLHttpRequest does not support mutual authentication. The AuthProxy plugin allows you to specify a certificate to include in an HTTPS request to identify the client to the server. This allows the server to verify the identity of the client. An example of where you might need mutual authentication is the onboarding process to register with an application, or, to access an OData producer. This occurs mostly in Business to Business (B2B) applications. This is different from most business to consumer (B2C) web sites where it is only the server that authenticates itself to the client with a certificate.

Adding and Removing the AuthProxy Plugin

The AuthProxy plugin is added and removed using the *Cordova CLI*.

To add the AuthProxy plugin to your project, use the following command:

```
cordova plugin add <path to directory containing Kapsel plugins>\authproxy
```

To remove the AuthProxy plugin from your project, use the following command:

```
cordova plugin rm com.sap.mp.cordova.plugins.authproxy
```

Classes

Name	Description
<i>sap.AuthProxy.CertificateFromFile</i> on page 129	Create certificate source description object for a certificate from a keystore file.
<i>sap.AuthProxy.CertificateFromLogonManager</i> on page 131	Create a certificate source description object for certificates from logon manager.
<i>sap.AuthProxy.CertificateFromStore</i> on page 132	Create a certificate source description object for certificates from the system keystore.

Members

Name	Description
<i>ERR_CERTIFICATE_ALIAS_NOT_FOUND</i> on page 132	Constant indicating the certificate with the given alias could not be found.
<i>ERR_CERTIFICATE_FILE_NOT_EXIST</i> on page 133	Constant indicating the certificate file could not be found.
<i>ERR_CERTIFICATE_INVALID_FILE_FORMAT</i> on page 133	Constant indicating incorrect certificate file format.
<i>ERR_CLIENT_CERTIFICATE_VALIDATION</i> on page 133	Constant indicating the provided certificate failed validation on the server side.
<i>ERR_FILE_CERTIFICATE_SOURCE_UNSUPPORTED</i> on page 133	Constant indicating the certificate from file is not supported on the current platform.

<i>ERR_GET_CERTIFICATE_FAILED</i> on page 134	Constant indicating failure in getting the certificate.
<i>ERR_HTTP_TIMEOUT</i> on page 134	Constant indicating timeout error while connecting to the server.
<i>ERR_INVALID_PARAMETER_VALUE</i> on page 134	Constant indicating the operation failed due to an invalid parameter (for example, a string was passed where a number was required).
<i>ERR_LOGON_MANAGER_CERTIFICATE_METHOD_NOT_AVAILABLE</i> on page 134	Constant indicating the logon manager certificate method is not available.
<i>ERR_LOGON_MANAGER_CORE_NOT_AVAILABLE</i> on page 135	Constant indicating the logon manager core library is not available.
<i>ERR_MISSING_PARAMETER</i> on page 135	Constant indicating the operation failed because of a missing parameter.
<i>ERR_NO_SUCH_ACTION</i> on page 135	Constant indicating there is no such Cordova action for the current service.
<i>ERR_SERVER_CERTIFICATE_VALIDATION</i> on page 136	Constant indicating the server certificate failed validation on the client side.
<i>ERR_SERVER_REQUEST_FAILED</i> on page 136	Constant indicating the server request failed.
<i>ERR_SYSTEM_CERTIFICATE_SOURCE_UNSUPPORTED</i> on page 136	Constant indicating the certificate from the system keystore is not supported on the current platform.
<i>ERR_UNKNOWN</i> on page 136	Constant indicating the operation failed with unknown error.

Methods

Name	Description
<i>deleteCertificateFromStore(successCB, [errorCB], certificateKey)</i> on page 137	Delete a cached certificate from the keychain.
<i>generateODataHttpClient()</i> on page 138	Generates an OData client that uses the Auth-Proxy plugin to make requests.
<i>get(url, header, successCB, errorCB, [user], [password], [timeout], [certSource])</i> on page 139	Send an HTTP(S) GET request to a remote server.

<i>sendRequest(method, url, header, requestBody, successCB, errorCallback, [user], [password], [timeout], [certSource])</i> on page 141	Send an HTTP(S) request to a remote server.
---	---

Type Definitions

Name	Description
<i>deleteCertificateSuccessCallback</i> on page 143	Callback function that is invoked upon successfully deleting a certificate from the store.
<i>errorCallback(errorObject)</i> on page 143	Callback function that is invoked in case of an error.
<i>successCallback(serverResponse)</i> on page 144	Callback function that is invoked upon a response from the server.

Source

authproxy.js, line 27 on page 146.

sap.AuthProxy.CertificateFromFile class

Create certificate source description object for a certificate from a keystore file.

The keystore file must be of type PKCS12 (usually a .p12 extension) since that is the only certificate file type that can contain a private key (a private key is needed to authenticate the client to the server). You might want to use this method if you know the desired certificate resides in a file on the filesystem.

Syntax

```
new CertificateFromFile( Path, Password, CertificateKey )
```

Parameters

Name	Type	Description
------	------	-------------

<i>Path</i>	string	<p>The Path of the keystore file.</p> <p>For iOS clients, it first tries to load the relative file path from the application's Documents folder. If it fails, it then tries to load the file path from application's main bundle. In addition, before trying to load the certificate from the file system, the iOS client first checks whether the specified certificate key already exists in the key store. If it does, it loads the existing certificate from key store, instead of loading the certificate from file system.</p> <p>For Android clients, the filepath is first treated as an absolute path. If the certificate is not found, then the filepath is treated as relative to the root of the sdcard.</p>
<i>Password</i>	string	The password of the keystore.
<i>CertificateKey</i>	string	A unique key (aka: alias) that is used to locate the certificate.

Example

```
// Create the certificate source description object.
var fileCert = new sap.AuthProxy.CertificateFromFile("directory/
certificateName.pl2", "certificatePassword", "certificateKey");
// callbacks
var successCB = function(serverResponse) {
    alert("Status: " + JSON.stringify(serverResponse.status));
    alert("Headers: " + JSON.stringify(serverResponse.headers));
    alert("Response: " + JSON.stringify(serverResponse.response));
}
var errorCallback = function(errorObject) {
    alert("Error making request: " + JSON.stringify(errorObject));
}
// Make the request with the certificate source description object.
sap.AuthProxy.sendRequest("POST", "https://hostname", headers, "THIS
IS THE BODY", successCB, errorCallback, null, null, 0, fileCert);
```

Source

authproxy.js, line 224 on page 154.

sap.AuthProxy.CertificateFromLogonManager class

Create a certificate source description object for certificates from logon manager.

Using the resulting certificate source description object on subsequent calls to `AuthProxy.sendRequest` or `AuthProxy.get` will cause `AuthProxy` to retrieve a certificate from Logon Manager to use for client authentication. The `appID` parameter is used to indicate which application's certificate to use.

Note that to use a certificate from Logon Manager, the application must have already registered with the server using a certificate from Afaria.

Syntax

```
new CertificateFromLogonManager( appID )
```

Parameters

Name	Type	Description
<i>appID</i>	string	application identifier

Example

```
// Create the certificate source description object.
var logonCert = new
sap.AuthProxy.CertificateFromLogonManager("applicationID");
// callbacks
var successCB = function(serverResponse) {
    alert("Status: " + JSON.stringify(serverResponse.status));
    alert("Headers: " + JSON.stringify(serverResponse.headers));
    alert("Response: " + JSON.stringify(serverResponse.response));
}
var errorCB = function(errorObject) {
    alert("Error making request: " + JSON.stringify(errorObject));
}
// Make the request with the certificate source description object.
sap.AuthProxy.sendRequest("POST", "https://hostname", headers, "THIS
IS THE BODY", successCB, errorCB, null, null, 0, logonCert);
```

Source

authproxy.js, line 281 on page 156.

sap.AuthProxy.CertificateFromStore class

Create a certificate source description object for certificates from the system keystore.

You might want to use a certificate from the system keystore if you know the user's device will have the desired certificate installed on it.

On Android, sending a request with a certificate from the system store results in UI being shown for the user to pick the certificate to use (the certificate with the alias matching the given CertificateKey is pre-selected).

Syntax

```
new CertificateFromStore( CertificateKey )
```

Parameters

Name	Type	Description
<i>CertificateKey</i>	string	A unique key (aka: alias) that is used to locate the certificate.

Example

```
// Create the certificate source description object.
var systemCert = new
sap.AuthProxy.CertificateFromStore("certificatekey");
// callbacks
var successCB = function(serverResponse) {
    alert("Status: " + JSON.stringify(serverResponse.status));
    alert("Headers: " + JSON.stringify(serverResponse.headers));
    alert("Response: " + JSON.stringify(serverResponse.response));
}
var errorCB = function(errorObject) {
    alert("Error making request: " + JSON.stringify(errorObject));
}
// Make the request with the certificate source description object.
sap.AuthProxy.sendRequest("POST", "https://hostname", headers, "THIS
IS THE BODY", successCB, errorCB, null, null, 0, systemCert);
```

Source

authproxy.js, line 253 on page 155.

ERR_CERTIFICATE_ALIAS_NOT_FOUND member

Constant indicating the certificate with the given alias could not be found.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

```
<constant> ERR_CERTIFICATE_ALIAS_NOT_FOUND : number
```


Source

authproxy.js, line 90 on page 149.

***ERR_CERTIFICATE_FILE_NOT_EXIST* member**

Constant indicating the certificate file could not be found.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

```
<constant> ERR_CERTIFICATE_FILE_NOT_EXIST : number
```

Source

authproxy.js, line 98 on page 149.

***ERR_CERTIFICATE_INVALID_FILE_FORMAT* member**

Constant indicating incorrect certificate file format.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

```
<constant> ERR_CERTIFICATE_INVALID_FILE_FORMAT : number
```

Source

authproxy.js, line 106 on page 149.

***ERR_CLIENT_CERTIFICATE_VALIDATION* member**

Constant indicating the provided certificate failed validation on the server side.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

```
<constant> ERR_CLIENT_CERTIFICATE_VALIDATION : number
```

Source

authproxy.js, line 122 on page 150.

***ERR_FILE_CERTIFICATE_SOURCE_UNSUPPORTED* member**

Constant indicating the certificate from file is not supported on the current platform.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

```
<constant> ERR_FILE_CERTIFICATE_SOURCE_UNSUPPORTED : number
```

Source

authproxy.js, line 74 on page 148.

ERR_GET_CERTIFICATE_FAILED member

Constant indicating failure in getting the certificate.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

<constant> ERR_GET_CERTIFICATE_FAILED : number

Source

authproxy.js, line 114 on page 149.

ERR_HTTP_TIMEOUT member

Constant indicating timeout error while connecting to the server.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

<constant> ERR_HTTP_TIMEOUT : number

Source

authproxy.js, line 164 on page 151.

ERR_INVALID_PARAMETER_VALUE member

Constant indicating the operation failed due to an invalid parameter (for example, a string was passed where a number was required).

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

<constant> ERR_INVALID_PARAMETER_VALUE : number

Source

authproxy.js, line 48 on page 147.

ERR_LOGON_MANAGER_CERTIFICATE_METHOD_NOT_AVAILABLE member

Constant indicating the logon manager certificate method is not available.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

<constant> ERR_LOGON_MANAGER_CERTIFICATE_METHOD_NOT_AVAILABLE :
number

Source

authproxy.js, line 156 on page 151.

***ERR_LOGON_MANAGER_CORE_NOT_AVAILABLE* member**

Constant indicating the logon manager core library is not available.

Getting this error code means you tried to use Logon plugin features (for example, a certificate from Logon) without adding the Logon plugin to the app. A possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

```
<constant> ERR_LOGON_MANAGER_CORE_NOT_AVAILABLE : number
```

Source

authproxy.js, line 148 on page 151.

***ERR_MISSING_PARAMETER* member**

Constant indicating the operation failed because of a missing parameter.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

```
<constant> ERR_MISSING_PARAMETER : number
```

Source

authproxy.js, line 56 on page 147.

***ERR_NO_SUCH_ACTION* member**

Constant indicating there is no such Cordova action for the current service.

When a Cordova plugin calls into native code it specifies an action to perform. If the action provided by the JavaScript is unknown to the native code this error occurs. This error should not occur as long as *authproxy.js* is unmodified. Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

```
<constant> ERR_NO_SUCH_ACTION : number
```

Source

authproxy.js, line 66 on page 148.

ERR_SERVER_CERTIFICATE_VALIDATION member

Constant indicating the server certificate failed validation on the client side.

This is likely because the server certificate is self-signed, or not signed by a well-known certificate authority. This constant is used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

<constant> ERR_SERVER_CERTIFICATE_VALIDATION : number

Source

authproxy.js, line 131 on page 150.

ERR_SERVER_REQUEST_FAILED member

Constant indicating the server request failed.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

<constant> ERR_SERVER_REQUEST_FAILED : number

Source

authproxy.js, line 139 on page 150.

ERR_SYSTEM_CERTIFICATE_SOURCE_UNSUPPORTED member

Constant indicating the certificate from the system keystore is not supported on the current platform.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

<constant> ERR_SYSTEM_CERTIFICATE_SOURCE_UNSUPPORTED : number

Source

authproxy.js, line 82 on page 148.

ERR_UNKNOWN member

Constant indicating the operation failed with unknown error.

Used as a possible value for the `errorCode` in *sap.AuthProxy~errorCallback* on page 143.

Syntax

<constant> ERR_UNKNOWN : number

Source

authproxy.js, line 40 on page 147.

deleteCertificateFromStore(successCB, [errorCB], certificateKey) method

Delete a cached certificate from the keychain.

iOS clients always checks the cached certificate first to see if it is available before loading the certificate from the file system. If the cached certificate is no longer valid, use this method to delete it from the keychain.

Only supported on iOS platform, NOT Android.

Syntax

`deleteCertificateFromStore(successCB, [errorCB], certificateKey)`

Parameters

Name	Type	Argument	Description
<i>successCB</i>	<i>sap.AuthProxy~deleteCertificateSuccess-Callback</i> on page 143		Callback method upon success.
<i>errorCB</i>	<i>sap.AuthProxy~error-Callback</i> on page 143	(optional)	Callback method upon failure.
<i>certificateKey</i>	string		The key of the certificate to be deleted.

Example

```
var successCB = function(){
    alert("certificate successfully deleted.");
}
var errorCB = function(error){
    alert("error deleting certificate: " + JSON.stringify(error));
}
sap.AuthProxy.deleteCertificateFromStore(successCB, errorCB,
"certificateKeyToDelete");
```

Source

authproxy.js, line 636 on page 170.

generateODataHttpClient() method

Generates an OData client that uses the AuthProxy plugin to make requests.

This is useful if you are using Datajs, but want to make use of the certificate features of AuthProxy. Datajs is a javascript library useful for accessing OData services. Datajs has a concept of an HttpClient, which does the work of making the request. This function generates an HttpClient that you can specify to Datajs so you can provide client certificates for requests. If you want to use the generated HTTP client for all future Datajs requests, you can do that by setting the OData.defaultHttpClient property to the return value of this function. Once that is done, then doing OData stuff with Datajs is almost exactly the same, but you can add a certificateSource to a request.

Syntax

```
generateODataHttpClient()
```

Example

```
OData.defaultHttpClient = sap.AuthProxy.generateODataHttpClient();  
  
// Using a certificate from file, for example.  
fileCert = new sap.AuthProxy.CertificateFromFile("mnt/sdcard/  
cert.pl2", "password", "certKey");  
  
// This is the same request object you would have created if you were  
// just using Datajs, but now  
// you can add the extra 'certificateSource' property.  
var createRequest = {  
    requestUri: "http://www.example.com/stuff/etc/example.svc",  
    certificateSource : fileCert,  
    user : "username",  
    password : "password",  
    method : "POST",  
    data:  
    {  
        Description: "Created Record",  
        CategoryName: "Created Category"  
    }  
}  
  
// Use Datajs to send the request.  
OData.request( createRequest, successCallback, failureCallback );
```

Source

authproxy.js, line 733 on page 173.

get(url, header, successCB, errorCallback, [user], [password], [timeout], [certSource])
method

Send an HTTP(S) GET request to a remote server.

This is a convenience function that simply calls *sap.AuthProxy#sendRequest* on page 141 with "GET" as the method and null for the request body. All given parameters are passed as-is to *sap.AuthProxy.sendRequest*. The success callback is invoked upon any response from the server. Even responses not generally considered to be successful (such as 404 or 500 status codes) will result in the success callback being invoked. The error callback is reserved for problems that prevent the AuthProxy from creating the request or contacting the server. It is, therefore, important to always check the status property on the object given to the success callback.

Syntax

```
get( url, header, successCB, errorCallback, [user], [password], [timeout], [certSource] )
{function}
```

Parameters

Name	Type	Argument	Description
<i>url</i>	string		The URL against which to make the request.
<i>header</i>	Object		HTTP header to send to the server. This is an Object. Can be null.
<i>successCB</i>	<i>sap.AuthProxy~successCallback</i> on page 144		Callback method invoked upon a response from the server.
<i>errorCB</i>	<i>sap.AuthProxy~errorCallback</i> on page 143		Callback method invoked in case of failure.
<i>user</i>	string	(optional)	User ID for basic authentication.
<i>password</i>	string	(optional)	User password for basic authentication.
<i>timeout</i>	number	(optional)	Timeout setting in seconds. Default value (0) means there is no timeout.

<i>certSource</i>	Object	(optional)	Certificate description object. It can be one of <i>sap.AuthProxy#CertificateFromFile</i> on page 129, <i>sap.AuthProxy#CertificateFromStore</i> on page 132, or <i>sap.AuthProxy#CertificateFromLogonManager</i> on page 131.
-------------------	--------	------------	--

Returns

A JavaScript function object to abort the operation. Calling the abort function results in neither the success or error callback being invoked for the original request (excepting the case where the success or error callback was invoked before calling the abort function). Note that the request itself cannot be unsent, and the server will still receive the request - the JavaScript will just not know the results of that request.

Type:

function

Example

```
var successCB = function(serverResponse) {
    alert("Status: " + JSON.stringify(serverResponse.status));
    alert("Headers: " + JSON.stringify(serverResponse.headers));
    if (serverResponse.responseText) {
        alert("Response: " +
JSON.stringify(serverResponse.responseText));
    }
}
var errorCallback = function(errorObject) {
    alert("Error making request: " + JSON.stringify(errorObject));
}
// To send a GET request to server, call the method
var abortFunction = sap.AuthProxy.get("http://www.example.com",
null, successCB, errorCallback);
// An example of aborting the request
abortFunction();
// To send a GET request to the server with headers, call the method
sap.AuthProxy.get("http://www.example.com", {HeaderName : "Header
value"}, successCB, errorCallback);
// To send a GET request to the server with basic authentication,
call the method
sap.AuthProxy.get("https://www.example.com", headers, successCB,
errorCB, "username", "password");
// To send a GET request to the server with mutual authentication,
call the method
```



```
sap.AuthProxy.get("https://www.example.com", headers, successCB,
errorCB, null, null, 0,
    new sap.AuthProxy.CertificateFromLogonManager("theAppId"));
```

Source

authproxy.js, line 616 on page 169.

sendRequest(method, url, header, requestBody, successCB, errorCB, [user], [password], [timeout], [certSource]) method

Send an HTTP(S) request to a remote server.

This function is the centerpiece of the AuthProxy plugin. It will handle mutual authentication if a certificate source is provided. The success callback is invoked upon any response from the server. Even responses not generally considered to be successful (such as 404 or 500 status codes) will result in the success callback being invoked. The error callback is reserved for problems that prevent the AuthProxy from creating the request or contacting the server. It is therefore important to always check the status property on the object given to the success callback.

Syntax

sendRequest(method, url, header, requestBody, successCB, errorCB, [user], [password], [timeout], [certSource]) {function}

Parameters

Name	Type	Argument	Description
<i>method</i>	string		Standard HTTP request method name.
<i>url</i>	string		The HTTP URL with format http(s)://[user:password]@host-name[:port]/path.
<i>header</i>	Object		HTTP header to send to the server. This is an Object. Can be null.
<i>requestBody</i>	string		Data to send to the server with the request. Can be null.
<i>successCB</i>	<i>sap.AuthProxy~successCallback</i> on page 144		Callback method invoked upon a response from the server.

<i>errorCB</i>	<i>sap.AuthProxy~error-Callback</i> on page 143		Callback method invoked in case of failure.
<i>user</i>	string	(optional)	User ID for basic authentication.
<i>password</i>	string	(optional)	User password for basic authentication.
<i>timeout</i>	number	(optional)	Timeout setting in seconds. Default value (0) means there is no timeout.
<i>certSource</i>	Object	(optional)	Certificate description object. It can be one of <i>sap.AuthProxy#CertificateFromFile</i> on page 129, <i>sap.AuthProxy#CertificateFromStore</i> on page 132, or <i>sap.AuthProxy#CertificateFromLogonManager</i> on page 131.

Returns

A JavaScript function object to abort the operation. Calling the abort function results in neither the success or error callback being invoked for the original request (excepting the case where the success or error callback was invoked before calling the abort function). Note that the request itself cannot be unsent, and the server will still receive the request - the JavaScript will just not know the results of that request.

Type:

function

Example

```
// callbacks
var successCB = function(serverResponse) {
    alert("Status: " + JSON.stringify(serverResponse.status));
    alert("Headers: " + JSON.stringify(serverResponse.headers));
    alert("Response: " + JSON.stringify(serverResponse.response));
}
var errorCB = function(errorObject) {
    alert("Error making request: " + JSON.stringify(errorObject));
}
// To send a post request to the server, call the method
```

```

var abortFunction = sap.AuthProxy.sendRequest("POST", "http://
www.google.com", null, "THIS IS THE BODY", successCB, errorCallback);
// An example of aborting the request
abortFunction();

// To send a post request to the server with headers, call the method
sap.AuthProxy.sendRequest("POST", url, {HeaderName : "Header
value"}, "THIS IS THE BODY", successCB, errorCallback);

// To send a post request to the server with basic authentication,
call the method
sap.AuthProxy.sendRequest("POST", url, headers, "THIS IS THE BODY",
successCB, errorCallback, "username", "password");

// To send a post request to the server with mutual authentication,
call the method
sap.AuthProxy.sendRequest("POST", "https://hostname", headers, "THIS
IS THE BODY", successCB, errorCallback, null,
null, 0, new
sap.AuthProxy.CertificateFromLogonManager("theAppId"));

```

Source

authproxy.js, line 466 on page 163.

***deleteCertificateSuccessCallback* type**

Callback function that is invoked upon successfully deleting a certificate from the store.

Syntax

`deleteCertificateSuccessCallback()`

Source

authproxy.js, line 819 on page 176.

***errorCallback(errorObject)* type**

Callback function that is invoked in case of an error.

Syntax

`errorCallback(errorObject)`

Parameters

Name	Type	Description

<i>errorObject</i>	Object	An object containing two properties: 'errorCode' and 'description.' The 'errorCode' property corresponds to one of the <i>sap.AuthProxy</i> on page 126 constants. The 'description' property is a string with more detailed information of what went wrong.
--------------------	--------	--

Example

```
function errorCallback(errCode) {
    //Set the default error message. Used if an invalid code is passed
to the
    //function (just in case) but also to cover the
    //sap.AuthProxy.ERR_UNKNOWN case as well.
    var msg = "Unkown Error";
    switch (errCode) {
        case sap.AuthProxy.ERR_INVALID_PARAMETER_VALUE:
            msg = "Invalid parameter passed to method";
            break;
        case sap.AuthProxy.ERR_MISSING_PARAMETER:
            msg = "A required parameter was missing";
            break;
        case sap.AuthProxy.ERR_HTTP_TIMEOUT:
            msg = "The request timed out";
            break;
    };
    //Write the error to the log
    console.error(msg);
    //Let the user know what happened
    navigator.notification.alert(msg, null, "AuthProxy Error", "OK");
};
```

Source

authproxy.js, line 815 on page 176.

successCallback(serverResponse) type

Callback function that is invoked upon a response from the server.

Syntax

successCallback(serverResponse)

Parameters

Name	Type	Description
------	------	-------------

<i>serverResponse</i>	Object	<p>An object containing the response from the server. Contains a 'headers' property, a 'status' property, and a 'responseText' property.</p> <p>'headers' is an object containing all the headers in the response.</p> <p>'status' is an integer corresponding to the HTTP status code of the response. It is important to check the status of the response, since this success callback is invoked upon any response from the server - including responses that are not normally thought of as successes (for example, the status code could be 404 or 500).</p> <p>'responseText' is a string containing the body of the response.</p>
-----------------------	--------	---

Source

authproxy.js, line 817 on page 176.

Source code***authproxy.js***

```

1 // ${project.version}
2 var exec = require('cordova/exec');
3
4 /**
5  * The AuthProxy plugin provides the ability to make HTTPS
  requests with mutual authentication.<br/>
6  * <br/>
7  * The regular XMLHttpRequest does not

```

Kapsel Development

```
8      * support mutual authentication. The AuthProxy plugin allows
you to specify a certificate to include in an HTTPS request
9      * to identify the client to the server. This allows the
server to verify the identity of the client. An example of where
you
10     * might need mutual authentication is the onboarding process
to register with an application, or, to access an
11     * OData producer. This occurs mostly in Business to Business
(B2B) applications. This is different from most business to
12     * consumer (B2C) Web sites, where it is only the server that
authenticates itself to the client with a certificate.<br/>
13     * <br/>
14     * <b>Adding and Removing the AuthProxy Plugin</b><br/>
15     * Add or remove the AuthProxy plugin using the
16     * <a href="http://cordova.apache.org/docs/en/edge/
guide_cli_index.md.html#The%20Command-line%20Interface">Cordova
CLI</a>.<br/>
17     * <br/>
18     * To add the AuthProxy plugin to your project, use the
following command:<br/>
19     * cordova plugin add <path to directory containing Kapsel
plugins>\authproxy<br/>
20     * <br/>
21     * To remove the AuthProxy plugin from your project, use the
following command:<br/>
22     * cordova plugin rm com.sap.mp.cordova.plugins.authproxy
23     * @namespace
24     * @alias AuthProxy
25     * @memberof sap
26     */
27     var AuthProxy = function () {};
28
29
30     /**
31     * Constant definitions for registration methods
32     */
33
```

```
34     /**
35      * Constant indicating the operation failed with unknown
error. Used as a possible value for the
36      * errorCode in {@link sap.AuthProxy~errorCallback}.
37      * @constant
38      * @type number
39      */
40     AuthProxy.prototype.ERR_UNKNOWN = -1;
41
42     /**
43      * Constant indicating the operation failed due to an invalid
parameter (for example, a string was passed where a number was
44      * required). Used as a possible value for the errorCode in
{@link sap.AuthProxy~errorCallback}.
45      * @constant
46      * @type number
47      */
48     AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE = -2;
49
50     /**
51      * Constant indicating the operation failed because of a
missing parameter. Used as a possible value for the
52      * errorCode in {@link sap.AuthProxy~errorCallback}.
53      * @constant
54      * @type number
55      */
56     AuthProxy.prototype.ERR_MISSING_PARAMETER = -3;
57
58     /**
59      * Constant indicating there is no such Cordova action for the
current service. When a Cordova plugin calls into native
60      * code it specifies an action to perform. If the action
provided by the JavaScript is unknown to the native code this
61      * error occurs. This error should not occur as long as
authproxy.js is unmodified. Used as a possible
```

Kapsel Development

```
62     * value for the errorCode in {@link
sap.AuthProxy~errorCallback}.
63     * @constant
64     * @type number
65     */
66     AuthProxy.prototype.ERR_NO_SUCH_ACTION = -100;
67
68     /**
69     * Constant indicating the certificate from file is not
supported on the current platform. Used as a possible value for the
70     * errorCode in {@link sap.AuthProxy~errorCallback}.
71     * @constant
72     * @type number
73     */
74     AuthProxy.prototype.ERR_FILE_CERTIFICATE_SOURCE_UNSUPPORTED
= -101;
75
76     /**
77     * Constant indicating the certificate from the system
keystore is not supported on the current platform. Used as a possible
value
78     * for the errorCode in {@link
sap.AuthProxy~errorCallback}.
79     * @constant
80     * @type number
81     */
82     AuthProxy.prototype.ERR_SYSTEM_CERTIFICATE_SOURCE_UNSUPPORTED =
-102;
83
84     /**
85     * Constant indicating the certificate with the given alias
could not be found. Used as a possible value for the
86     * errorCode in {@link sap.AuthProxy~errorCallback}.
87     * @constant
88     * @type number
```



```
89     */
90     AuthProxy.prototype.ERR_CERTIFICATE_ALIAS_NOT_FOUND = -104;
91
92     /**
93     * Constant indicating the certificate file could not be
94     * found. Used as a possible value for the
95     * errorCode in {@link sap.AuthProxy~errorCallback}.
96     * @constant
97     * @type number
98     */
99
100    /**
101    * Constant indicating incorrect certificate file format.
102    * Used as a possible value for the
103    * errorCode in {@link sap.AuthProxy~errorCallback}.
104    * @constant
105    * @type number
106    */
107
108    AuthProxy.prototype.ERR_CERTIFICATE_INVALID_FILE_FORMAT =
-106;
109
110    /**
111    * Constant indicating failure in getting the certificate.
112    * Used as a possible value for the
113    * errorCode in {@link sap.AuthProxy~errorCallback}.
114    * @constant
115    * @type number
116    */
117
118    AuthProxy.prototype.ERR_GET_CERTIFICATE_FAILED = -107;
119
120    /**
121    * Constant indicating the provided certificate failed
122    * validation on the server side. Used as a possible value for the
```

```
118     * errorCode in {@link sap.AuthProxy~errorCallback}.
119     * @constant
120     * @type number
121     */
122     AuthProxy.prototype.ERR_CLIENT_CERTIFICATE_VALIDATION =
-108;
123
124     /**
125     * Constant indicating the server certificate failed
validation on the client side. This is likely because the server
certificate
126     * is self-signed, or not signed by a well-known certificate
authority. This constant is used as a possible value for the
127     * errorCode in {@link sap.AuthProxy~errorCallback}.
128     * @constant
129     * @type number
130     */
131     AuthProxy.prototype.ERR_SERVER_CERTIFICATE_VALIDATION =
-109;
132
133     /**
134     * Constant indicating the server request failed. Used as a
possible value for the
135     * errorCode in {@link sap.AuthProxy~errorCallback}.
136     * @constant
137     * @type number
138     */
139     AuthProxy.prototype.ERR_SERVER_REQUEST_FAILED = -110;
140
141     /**
142     * Constant indicating the Logon Manager core library is not
available. Getting this error code means you tried
143     * to use Logon plugin features (for example, a certificate
from Logon) without adding the Logon plugin to the app.
144     * A possible value for the errorCode in {@link
sap.AuthProxy~errorCallback}.
```

```
145     * @constant
146     * @type number
147     */
148     AuthProxy.prototype.ERR_LOGON_MANAGER_CORE_NOT_AVAILABLE =
-111;
149
150     /**
151     * Constant indicating the Logon Manager certificate method is
not available. Used as a possible value for the
152     * errorCode in {@link sap.AuthProxy~errorCallback}.
153     * @constant
154     * @type number
155     */
156
AuthProxy.prototype.ERR_LOGON_MANAGER_CERTIFICATE_METHOD_NOT_AVAILA
BLE = -112;
157
158     /**
159     * Constant indicating timeout error while connecting to the
server. Used as a possible value for the
160     * errorCode in {@link sap.AuthProxy~errorCallback}.
161     * @constant
162     * @type number
163     */
164     AuthProxy.prototype.ERR_HTTP_TIMEOUT = -120;
165
166     /**
167     * Constant indicating timeout error while connecting to the
server. Used as a possible value for the
168     * errorCode in {@link sap.AuthProxy~errorCallback}.
169     * @constant
170     * @type number
171     */
172
```

Kapsel Development

```
173     /**
174     * Constant indicating a missing required parameter message.
175     * Used as a possible value for the description
176     * in (@link sap.AuthProxy~errorCallback).
177     * @constant
178     * @type string
179     * @private
180     */
181     AuthProxy.prototype.MSG_MISSING_PARAMETER = "Missing a
182     required parameter: ";
183
184     /**
185     * Constant indicating invalid parameter value message. Used
186     * as a possible value for the description
187     * in (@link sap.AuthProxy~errorCallback).
188     * @constant
189     * @type string
190     * @private
191     */
192     AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE = "Invalid
193     Parameter Value for parameter: ";
194
195     /**
196     * Create certificate source description object for a
197     * certificate from a keystore file. The keystore file must be of type
198     * PKCS12
199     * (usually a .p12 extension) since that is the only
200     * certificate file type that can contain a private key (a private key
201     * is needed
202     * to authenticate the client to the server). You might want
203     * to use this method if you know the desired certificate resides in a
204     * file on the filesystem.
205     * @class
206     * @param {string} Path The Path of the keystore file.<br/>For
207     * iOS clients, it first tries to load the
208     * relative file path from the application's
209     * Documents folder. If it fails, it then tries
```

```

199      *          to load the file path from the application's
main bundle. In addition, before trying

200      *          to load the certificate from the file system,
the iOS client first checks whether the

201      *          specified certificate key already exists in
the key store. If it does, it loads

202      *          the existing certificate from the key store,
instead of loading the certificate from the

203      *          file system.<br/>

204      *          For Android clients, the file path is first
treated as an absolute path. If the certificate

205      *          is not found, the file path is treated as
relative to the root of the SD card.

206      * @param {string} Password The password of the keystore.

207      * @param {string} CertificateKey A unique key (alias) that is
used to locate the certificate.

208      * @example

209      * // Create the certificate source description object.

210      * var fileCert = new
sap.AuthProxy.CertificateFromFile("directory/certificateName.p12",
"certificatePassword", "certificateKey");

211      * // callbacks

212      * var successCB = function(serverResponse) {

213      *     alert("Status: " +
JSON.stringify(serverResponse.status));

214      *     alert("Headers: " +
JSON.stringify(serverResponse.headers));

215      *     alert("Response: " +
JSON.stringify(serverResponse.response));

216      * }

217      * var errorCallback = function(errorObject) {

218      *     alert("Error making request: " +
JSON.stringify(errorObject));

219      * }

220      * // Make the request with the certificate source description
object.

221      * sap.AuthProxy.sendRequest("POST", "https://hostname",
headers, "THIS IS THE BODY", successCB, errorCallback, null, 0,
fileCert);

```

```
222     *
223     */
224     AuthProxy.prototype.CertificateFromFile = function (Path,
225     Password, CertificateKey) {
226         this.Source = "FILE";
227         this.Path = Path;
228         this.Password = Password;
229         this.CertificateKey = CertificateKey;
230     };
231     /**
232     * Create a certificate source description object for
233     * certificates from the system keystore. You might want to use a
234     * certificate
235     * from the system keystore if you know the user's device will
236     * have the desired certificate installed on it.<br/>
237     * On Android, sending a request with a certificate from the
238     * system store results in UI being shown, where the user can pick
239     * the certificate to use (the certificate with the alias
240     * matching the given CertificateKey is pre-selected).
241     * @class
242     * @param {string} CertificateKey A unique key (alias) that is
243     * used to locate the certificate.
244     * @example
245     * // Create the certificate source description object.
246     * var systemCert = new
247     sap.AuthProxy.CertificateFromStore("certificatekey");
248     * // callbacks
249     * var successCB = function(serverResponse){
250     *     alert("Status: " +
251     JSON.stringify(serverResponse.status));
252     *     alert("Headers: " +
253     JSON.stringify(serverResponse.headers));
254     *     alert("Response: " +
255     JSON.stringify(serverResponse.response));
256     * }
257     * var errorCallback = function(errorObject){
```

```

248     *     alert("Error making request: " +
JSON.stringify(errorObject));
249     * }

250     * // Make the request with the certificate source description
object.

251     * sap.AuthProxy.sendRequest("POST", "https://hostname",
headers, "THIS IS THE BODY", successCB, errorCallback, null, null, 0,
systemCert);
252     */

253     AuthProxy.prototype.CertificateFromStore = function
(CertificateKey) {
254         this.Source = "SYSTEM";
255         this.CertificateKey = CertificateKey;
256     };
257
258
259     /**
260     * Create a certificate source description object for
certificates from Logon Manager. Using the resulting certificate
source description
261     * object on subsequent calls to AuthProxy.sendRequest or
AuthProxy.get causes AuthProxy to retrieve a certificate from Logon
Manager
262     * to use for client authentication. The appID parameter is
used to indicate which application's certificate to use.<br/>
263     * Note: To use a certificate from Logon Manager, the
application must have already registered with the server using a
certificate from Afaria.
264     * @class
265     * @param {string} appID application identifier
266     * @example
267     * // Create the certificate source description object.
268     * var logonCert = new
sap.AuthProxy.CertificateFromLogonManager("applicationID");
269     * // callbacks
270     * var successCB = function(serverResponse){
271     *     alert("Status: " +
JSON.stringify(serverResponse.status));

```

Kapsel Development

```
272     *     alert("Headers: " +
JSON.stringify(serverResponse.headers));

273     *     alert("Response: " +
JSON.stringify(serverResponse.response));

274     * }

275     * var errorCallback = function(errorObject){

276     *     alert("Error making request: " +
JSON.stringify(errorObject));

277     * }

278     * // Make the request with the certificate source description
object.

279     * sap.AuthProxy.sendRequest("POST", "https://hostname",
headers, "THIS IS THE BODY", successCB, errorCallback, null, null, 0,
logonCert);

280     */

281     AuthProxy.prototype.CertificateFromLogonManager = function
(appID) {

282         this.Source = "LOGON";

283         this.AppID = appID;

284     };

285

286

287     /**

288     * Verifies that a certificate source description object
(created with {@link sap.AuthProxy#CertificateFromFile},

289     * {@link sap.AuthProxy#CertificateFromStore}, or {@link
sap.AuthProxy#CertificateFromLogonManager}) has all the required
fields and that the values

290     * for those fields are the correct type. This function
verifies only the certificate description object, not the certificate
itself. So, for example,

291     * if the certificate source description object was created
with {@link sap.AuthProxy#CertificateFromFile} and has a string for
the file path and a

292     * string for the key/alias, <b>this function considers it
valid even if no certificate actually exists on the file system</b>.
If the certificate

293     * source description object is valid but the certificate
itself is not, then an error occurs during the call to {@link
sap.AuthProxy#get} or
```



```
294     * {@link sap.AuthProxy#sendRequest}.
295     * @param {object} certSource The certificate source
object.
296     * @param {sap.AuthProxy~errorCallback} errorCB The error
callback invoked if the certificate source is not valid. Will have
an object with 'errorCode'
297     * and 'description' properties.
298     * @example
299     * var notValidCert = {};
300     * var errorCallback = function(error){
301     *     alert("certificate not valid!\nError code: " +
error.errorCode + "\ndescription: " + error.description);
302     * }
303     * var isCertValid =
sap.AuthProxy.validateCertSource(notValidCert, errorCallback);
304     * if( isCertValid ){
305     *     // do stuff with the valid certificate source
description object
306     * } else {
307     *     // at this point we know the cert is not valid, and the
error callback is invoked with extra information.
308     * }
309     *
310     *
311     * Developers are not expected to call this function.
312     * @private
313     */
314     AuthProxy.prototype.validateCertSource = function
(certSource, errorCB) {
315         if (!certSource) {
316             // The certificate is not present, so just ignore
it.
317             return true;
318         }
319
320         // errorCB required.
```

```
321         // First check this one. We may need it to return
errors
322         if (errorCB && (typeof errorCB !== "function")) {
323             console.log("AuthProxy Error: errorCB is not a
function");
324             return false;
325         }
326
327         try {
328             // First check whether it is an object
329             if (typeof certSource !== "object") {
330                 errorCB({
331                     errorCode:
AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
332                     description:
AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE + "certSource"
333                 });
334                 return false;
335             }
336
337             if (certSource.Source === "FILE") {
338                 if (!certSource.Path) {
339                     errorCB({
340                         errorCode:
AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
341                         description:
AuthProxy.prototype.MSG_MISSING_PARAMETER + "keystore path"
342                     });
343                     return false;
344                 }
345
346                 if (typeof certSource.Path !== "string") {
347                     errorCB({
348                         errorCode:
AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
```

```
349             description:
AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE + "keystore path"
350         });
351         return false;
352     }
353
354     if (!certSource.CertificateKey) {
355         errorCallback({
356             errorCode:
AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
357             description:
AuthProxy.prototype.MSG_MISSING_PARAMETER + "certificate key"
358         });
359         return false;
360     }
361
362     if (typeof certSource.CertificateKey !== "string")
{
363         errorCallback({
364             errorCode:
AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
365             description:
AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE + "certificate key"
366         });
367         return false;
368     }
369     } else if (certSource.Source === "SYSTEM") {
370         if (!certSource.CertificateKey) {
371             errorCallback({
372                 errorCode:
AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
373                 description:
AuthProxy.prototype.MSG_MISSING_PARAMETER + "certificate key"
374             });
375             return false;
```

```
376         }
377
378         if (typeof certSource.CertificateKey !== "string")
379         {
380             errorCallback({
381                 errorCode:
382                 AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
383                 description:
384                 AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE + "certificate key"
385             });
386             return false;
387         }
388     } else if (certSource.Source === "LOGON") {
389         if (!certSource.AppID) {
390             errorCallback({
391                 errorCode:
392                 AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
393                 description:
394                 AuthProxy.prototype.MSG_MISSING_PARAMETER + "AppID"
395             });
396             return false;
397         }
398     }
399     if (typeof certSource.AppID !== "string") {
400         errorCallback({
401             errorCode:
402             AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
```

```

403         errorCode:
AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
404         description:
AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE + "certSource"
405     });
406     return false;
407 }
408
409     return true;
410 } catch (ex) {
411     errorCallback({
412         errorCode:
AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
413         description:
AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE + "certSource"
414     });
415 }
416 };
417
418
419 /**
420  * Send an HTTP(S) request to a remote server. This function
is the centerpiece of the AuthProxy plugin. It handles
421  * mutual authentication if a certificate source is
provided.
422  * The success callback is invoked upon any response from the
server, even responses not generally considered to be
423  * successful (such as 404 or 500 status codes) result in the
success callback being invoked. The error callback
424  * is reserved for problems that prevent the AuthProxy from
creating the request or contacting the server. It is, therefore,
425  * important to always check the status property on the object
given to the success callback.
426  * @param {string} method Standard HTTP request method
name.
427  * @param {string} url The HTTP URL with format http(s)://
[user:password]@hostname[:port]/path.

```

```
428      * @param {Object} header HTTP header to send to the server.  
This is an Object. Can be null.  
429      * @param {string} requestBody Data to send to the server with  
the request. Can be null.  
430      * @param {sap.AuthProxy~successCallback} successCB Callback  
method invoked upon a response from the server.  
431      * @param {sap.AuthProxy~errorCallback} errorCB Callback  
method invoked in case of failure.  
432      * @param {string} [user] User ID for basic authentication.  
433      * @param {string} [password] User password for basic  
authentication.  
434      * @param {number} [timeout] Timeout setting in seconds.  
Default value (0) means there is no timeout.  
435      * @param {Object} [certSource] Certificate description  
object. It can be one of {@link sap.AuthProxy#CertificateFromFile},  
436      * {@link sap.AuthProxy#CertificateFromStore}, or {@link  
sap.AuthProxy#CertificateFromLogonManager}.  
437      * @return {function} A JavaScript function object to abort  
the operation. Calling the abort function results in neither the  
success or error  
438      * callback being invoked for the original request (excepting  
the case where the success or error callback was invoked before  
calling the  
439      * abort function). Note: The request itself cannot be  
unsent, and the server will still receive the request, but the  
JavaScript will  
440      * not know the results of that request.  
441      * @example  
442      * // callbacks  
443      * var successCB = function(serverResponse){  
444      *     alert("Status: " +  
JSON.stringify(serverResponse.status));  
445      *     alert("Headers: " +  
JSON.stringify(serverResponse.headers));  
446      *     alert("Response: " +  
JSON.stringify(serverResponse.response));  
447      * }  
448      * var errorCallback = function(errorObject){  
449      *     alert("Error making request: " +  
JSON.stringify(errorObject));
```

```
450     * }
451     * // To send a post request to the server, call the method
452     * var abortFunction = sap.AuthProxy.sendRequest("POST",
453     "http://www.google.com", null, "THIS IS THE BODY", successCB,
454     errorCallback);
455     * // An example of aborting the request
456     * abortFunction();
457     *
458     * // To send a post request to the server with headers, call
459     * the method
460     * sap.AuthProxy.sendRequest("POST", url, {HeaderName :
461     "Header value"}, "THIS IS THE BODY", successCB, errorCallback);
462     *
463     * // To send a post request to the server with basic
464     * authentication, call the method
465     * sap.AuthProxy.sendRequest("POST", url, headers, "THIS IS
466     * THE BODY", successCB, errorCallback, "username", "password");
467     *
468     * // To send a post request to the server with mutual
469     * authentication, call the method
470     * sap.AuthProxy.sendRequest("POST", "https://hostname",
471     * headers, "THIS IS THE BODY", successCB, errorCallback, null,
472     * null, 0, new
473     * sap.AuthProxy.CertificateFromLogonManager("theAppId"));
474     */
475
476     AuthProxy.prototype.sendRequest = function (method, url,
477     header, requestBody, successCB, errorCallback, user, password, timeout,
478     certSource) {
479
480         // errorCallback required.
481         // First check this one. We may need it to return
482         // errors
483         if (!errorCB || (typeof errorCallback !== "function")) {
484             console.log("AuthProxy Error: errorCallback is not a
485             function");
486             // if error callback is invalid, throw an exception to
487             // notify the caller
```

```
473         throw new Error("AuthProxy Error: errorCallback is not a
function");
474     }
475
476     // method required
477     if (!method) {
478         console.log("AuthProxy Error: method is required");
479         errorCallback({
480             errorCode:
AuthProxy.prototype.ERR_MISSING_PARAMETER,
481             description:
AuthProxy.prototype.MSG_MISSING_PARAMETER + "method"
482         });
483         return;
484     }
485
486
487     // We only support GET, POST, HEAD, PUT, DELETE method
488     if (method !== "GET" && method !== "POST" && method !==
"HEAD" && method !== "PUT" && method !== "DELETE") {
489         console.log("Invalid Parameter Value for parameter: "
+ method);
490         errorCallback({
491             errorCode:
AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
492             description:
AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE + "method"
493         });
494         return;
495     }
496
497
498     // url required
499     if (!url) {
500         console.log("AuthProxy Error: url is required");
```



```
501         errorCallback({
502             errorCode:
AuthProxy.prototype.ERR_MISSING_PARAMETER,
503             description:
AuthProxy.prototype.MSG_MISSING_PARAMETER + "url"
504         });
505         return;
506     }
507
508
509     // successCB required
510     if (!successCB) {
511         console.log("AuthProxy Error: successCB is
required");
512         errorCallback({
513             errorCode:
AuthProxy.prototype.ERR_MISSING_PARAMETER,
514             description:
AuthProxy.prototype.MSG_MISSING_PARAMETER + "successCB"
515         });
516         return;
517     }
518
519
520     if (typeof successCB !== "function") {
521         console.log("AuthProxy Error: successCB is not a
function");
522         errorCallback({
523             errorCode:
AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
524             description:
AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE + "successCB"
525         });
526         return;
527     }
```

```
528
529
530     if (user && typeof user !== "string") {
531         errorCallback({
532             errorCode:
533             AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
534             description:
535             AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE + "user"
536         });
537     };
538
539     return;
540 }
541
542     if (password && typeof password !== "string") {
543         errorCallback({
544             errorCode:
545             AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
546             description:
547             AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE + "password"
548         });
549     };
550
551     return;
552 }
553
554     if (timeout && typeof timeout !== "number") {
555         errorCallback({
556             errorCode:
557             AuthProxy.prototype.ERR_INVALID_PARAMETER_VALUE,
558             description:
559             AuthProxy.prototype.MSG_INVALID_PARAMETER_VALUE + "timeout"
560         });
561     };
562
563     return;
564 }
565
```

```
556         if (!this.validateCertSource(certSource, errorCallback)) {
557             return;
558         }
559
560
561         try {
562             var client = new Client(method, url, header,
requestBody, successCB, errorCallback, user, password, timeout,
certSource);
563             return client.send();
564         } catch (ex) {
565             errorCallback({
566                 errorCode: AuthProxy.prototype.ERR_UNKNOWN,
567                 description: ex.message
568             });
569         }
570
571     };
572
573     /**
574     * Send an HTTP(S) GET request to a remote server. This is a
convenience function that simply calls {@link
sap.AuthProxy#sendRequest}
575     * with "GET" as the method and null for the request body.
All given parameters are passed as-is to sap.AuthProxy.sendRequest.
576     * The success callback is invoked upon any response from the
server, even responses not generally considered to be
577     * successful (such as 404 or 500 status codes) result in the
success callback being invoked. The error callback
578     * is reserved for problems that prevent the AuthProxy from
creating the request or contacting the server. It is, therefore,
579     * important to always check the status property on the object
given to the success callback.
580     * @param {string} url The URL against which to make the
request.
581     * @param {Object} header HTTP header to send to the server.
This is an Object. Can be null.
```

```
582     * @param {sap.AuthProxy~successCallback} successCB Callback
method invoked upon a response from the server.

583     * @param {sap.AuthProxy~errorCallback} errorCB Callback
method invoked in case of failure.

584     * @param {string} [user] User ID for basic authentication.

585     * @param {string} [password] User password for basic
authentication.

586     * @param {number} [timeout] Timeout setting in seconds.
Default value (0) means there is no timeout.

587     * @param {Object} [certSource] Certificate description
object. It can be one of {@link sap.AuthProxy#CertificateFromFile},
588     * {@link sap.AuthProxy#CertificateFromStore}, or {@link
sap.AuthProxy#CertificateFromLogonManager}.

589     * @return {function} A JavaScript function object to abort
the operation. Calling the abort function results in neither the
success or error

590     * callback being invoked for the original request (excepting
the case where the success or error callback was invoked before
calling the

591     * abort function). Note: The request itself cannot be
unsent, and the server will still receive the request, but the
JavaScript will

592     * not know the results of that request.

593     * @example

594     * var successCB = function(serverResponse){
595     *     alert("Status: " +
JSON.stringify(serverResponse.status));
596     *     alert("Headers: " +
JSON.stringify(serverResponse.headers));
597     *     if (serverResponse.responseText){
598     *         alert("Response: " +
JSON.stringify(serverResponse.responseText));
599     *     }
600     * }

601     * var errorCallback = function(errorObject){
602     *     alert("Error making request: " +
JSON.stringify(errorObject));
603     * }

604     * // To send a GET request to server, call the method
```

```

605     * var abortFunction = sap.AuthProxy.get("http://
www.example.com", null, successCB, errorCallback);

606     * // An example of aborting the request

607     * abortFunction();

608     * // To send a GET request to the server with headers, call
the method

609     * sap.AuthProxy.get("http://www.example.com", {HeaderName :
"Header value"}, successCB, errorCallback);

610     * // To send a GET request to the server with basic
authentication, call the method

611     * sap.AuthProxy.get("https://www.example.com", headers,
successCB, errorCallback, "username", "password");

612     * // To send a GET request to the server with mutual
authentication, call the method

613     * sap.AuthProxy.get("https://www.example.com", headers,
successCB, errorCallback, null, null, 0,

614     *     new
sap.AuthProxy.CertificateFromLogonManager("theAppId"));

615     */

616     AuthProxy.prototype.get = function (url, header, successCB,
errorCB, user, password, timeout, certSource) {

617         return this.sendRequest("GET", url, header, null,
successCB, errorCallback, user, password, timeout, certSource);

618     };

619

620     /**

621     * Delete a cached certificate from the keychain. iOS clients
always check the cached certificate first to see if it is available
before

622     * loading the certificate from the file system. If the cached
certificate is no longer valid, use this method to delete it from the
keychain.

623     * <br/><b>This function is supported only on iOS.</b>

624     * @param {sap.AuthProxy~deleteCertificateSuccessCallback}
successCB Callback method upon success.

625     * @param {sap.AuthProxy~errorCallback} [errorCB] Callback
method upon failure.

626     * @param {string} certificateKey The key of the certificate
to delete.

```

Kapsel Development

```
627     * @example
628     * var successCB = function(){
629     *     alert("certificate successfully deleted.");
630     * }
631     * var errorCallback = function(error){
632     *     alert("error deleting certificate: " +
633     *         JSON.stringify(error));
634     * }
635     */
636     AuthProxy.prototype.deleteCertificateFromStore = function
637     (successCB, errorCallback, certificateKey) {
638         cordova.exec(successCB, errorCallback, "AuthProxy",
639         "deleteCertificateFromStore", [certificateKey]);
640     };
641     /**
642     * @private
643     */
644     var Client = function (method, url, header, requestBody,
645     successCB, errorCallback, user, password, timeout, certSource) {
646         //ios plugin parameter does not support object type,
647         //convert Header and CertSource to JSON string
648         if (device.platform === "iOS" || (device.platform &&
649         device.platform.indexOf("iP") === 0)) {
650             if (header) {
651                 header = JSON.stringify(header);
652             }
653             if (certSource) {
654                 certSource = JSON.stringify(certSource);
655             }
656         }
657     };
658 }
```

```
655     this.Method = method;
656     this.Url = url;
657     this.Header = header;
658     this.RequestBody = requestBody;
659     this.SuccessCB = successCB;
660     this.ErrorCB = errorCB;
661     this.User = user;
662     this.Password = password;
663     this.Timeout = timeout;
664     this.CertSource = certSource;
665     this.IsAbort = false;
666
667     this.abort = function () {
668         this.IsAbort = true;
669     };
670
671
672     this.send = function () {
673
674         var args = [this.Method, this.Url, this.Header,
675 this.RequestBody, this.User, this.Password, this.Timeout,
676 this.CertSource];
677
678         var me = this;
679
680         var successCallBack = function (data) {
681             if (me.IsAbort === true) {
682                 return;
683             }
684             successCB(data);
685         };
686     };
687 }
```

```
686         var errorCallback = function (data) {
687             if (me.IsAbort === true) {
688                 return;
689             }
690
691             errorCallback(data);
692         };
693
694         exec(successCallback, errorCallback, "AuthProxy",
695             "sendRequest", args);
696
697         return this.abort;
698     };
699
700     /**
701     * Generates an OData client that uses the AuthProxy plugin to
702     * make requests. This is useful if you are using Datajs, but want
703     * to make use of the certificate features of AuthProxy.
704     * Datajs is a JavaScript library useful for accessing OData services.
705     * Datajs has a concept of an HttpClient, which does the work
706     * of making the request. This function generates an HttpClient that
707     * you can specify to Datajs so you can provide client
708     * certificates for requests. If you want to use the generated HTTP
709     * client
710     * for all future Datajs requests, you can do that by setting
711     * the OData.defaultHttpClient property to the return value of this
712     * function. Once that is done, then doing OData stuff with
713     * Datajs is almost exactly the same, but you can add a
714     * certificateSource to a request.
715     * @example
716     * OData.defaultHttpClient =
717     * sap.AuthProxy.generateODataHttpClient();
718     *
719     * // Using a certificate from file, for example.
```



```
712     * fileCert = new sap.AuthProxy.CertificateFromFile("mnt/
sdcard/cert.pl2", "password", "certKey");
713     *
714     * // This is the same request object you would have created
if you were just using Datajs, but now
715     * // you can add the extra 'certificateSource' property.
716     * var createRequest = {
717     *     requestUri: "http://www.example.com/stuff/etc/
example.svc",
718     *     certificateSource : fileCert,
719     *     user : "username",
720     *     password : "password",
721     *     method : "POST",
722     *     data:
723     *     {
724     *         Description: "Created Record",
725     *         CategoryName: "Created Category"
726     *     }
727     * }
728     *
729     * // Use Datajs to send the request.
730     * OData.request( createRequest, successCallback,
failureCallback );
731     *
732     */
733     AuthProxy.prototype.generateODataHttpClient = function () {
734         var httpClient = {
735             request: function (request, success, error) {
736                 var url, requestHeaders, requestBody, statusCode,
statusText, responseHeaders;
737                 var responseBody, requestTimeout, requestUserName,
requestPassword, requestCertificate;
738                 var client, result;
739
```

```
740         url = request.requestUri;
741         requestHeaders = request.headers;
742         requestBody = request.body;
743
744         var successCB = function (data) {
745             var response = {
746                 requestUri: url,
747                 statusCode: data.status,
748                 statusText: data.statusText,
749                 headers: data.headers,
750                 body: (data.responseText ?
data.responseText : data.responseBase64)
751             };
752
753             if (response.statusCode >= 200 &&
response.statusCode <= 299) {
754                 if (success) {
755                     success(response);
756                 }
757             } else {
758                 if (error) {
759                     error({
760                         message: "HTTP request failed",
761                         request: request,
762                         response: response
763                     });
764                 }
765             }
766         };
767
768         var errorCallback = function (data) {
769             if (error) {
770                 error({
```

```
771             message: data
772         });
773     }
774 };
775
776     if (request.timeoutMS) {
777         requestTimeout = request.timeoutMS / 1000;
778     }
779
780     if (request.certificateSource) {
781         requestCertificate =
782 request.certificateSource;
783     }
784
785     if (request.user) {
786         requestUserName = request.user;
787     }
788
789     if (request.password) {
790         requestPassword = request.password;
791     }
792
793     client =
794 AuthProxy.prototype.sendRequest(request.method || "GET", url,
795 requestHeaders, requestBody, successCB, errorCallback, requestUserName,
796 requestPassword, requestTimeout, requestCertificate);
797
798     result = {};
799     result.abort = function () {
800         client.abort();
801     }
802
803     if (error) {
804         error({
805             message: "Request aborted"
806         });
807     }
808 }
```

```
801         });
802     }
803     };
804     return result;
805 }
806 };
807 return httpClient;
808 };
809
810 var AuthProxyPlugin = new AuthProxy();
811
812 module.exports = AuthProxyPlugin;
813
814
815 /**
816  * Callback function that is invoked in case of an error.
817  *
818  * @callback sap.AuthProxy~errorCallback
819  *
820  * @param {Object} errorObject An object containing two
821  * properties: 'errorCode' and 'description.'
822  * The 'errorCode' property corresponds to one of the {@link
823  * sap.AuthProxy} constants. The 'description'
824  * property is a string with more detailed information of what
825  * went wrong.
826  *
827  * @example
828  * function errorCallback(errCode) {
829  *     //Set the default error message. Used if an invalid code
830  *     is passed to the
831  *     //function (just in case) but also to cover the
832  *     //sap.AuthProxy.ERR_UNKNOWN case as well.
833  *     var msg = "Unkown Error";
834  *     switch (errCode) {
```

```

831      *      case sap.AuthProxy.ERR_INVALID_PARAMETER_VALUE:
832      *          msg = "Invalid parameter passed to method";
833      *          break;
834      *      case sap.AuthProxy.ERR_MISSING_PARAMETER:
835      *          msg = "A required parameter was missing";
836      *          break;
837      *      case sap.AuthProxy.ERR_HTTP_TIMEOUT:
838      *          msg = "The request timed out";
839      *          break;
840      *      };
841      *      //Write the error to the log
842      *      console.error(msg);
843      *      //Let the user know what happened
844      *      navigator.notification.alert(msg, null, "AuthProxy
Error", "OK");
845      *  };
846      */
847
848      /**
849      * Callback function that is invoked upon a response from the
server.
850      *
851      * @callback sap.AuthProxy~successCallback
852      *
853      * @param {Object} serverResponse An object containing the
response from the server. Contains a 'headers' property,
854      * a 'status' property, and a 'responseText' property.<br/>
855      * 'headers' is an object containing all the headers in the
response.<br/>
856      * 'status' is an integer corresponding to the HTTP status
code of the response. It is important to check the status of
857      * the response, since <b>this success callback is invoked
upon any response from the server</b> - including responses that
are

```

```
858      * not normally thought of as successes (for example, the
status code could be 404 or 500).<br/>
859      * 'responseText' is a string containing the body of the
response.
860      */
861
862     /**
863      * Callback function that is invoked upon successfully
deleting a certificate from the store.
864      *
865      * @callback sap.AuthProxy~deleteCertificateSuccessCallback
866      */
```

Using the Logger Plugin

The Logger plugin includes client-side APIs that you can use for logging the activities of your application.

Logger Plugin Overview

The Logger plugin allows you to log information to trace bugs or other issues in your application for analysis.

Note: To upload log files successfully with the Logger plugin, these conditions must be met:

- In Management Cockpit, the **Log Upload** check box must be selected.
 - The `sap.Logger.upload()` must be called.
-

With the Logger plugin, you can enable an application to write log entries that can then be automatically uploaded to SAP Mobile Platform Server for analysis by using the `sap.Logger.upload()` method. If you add the Settings plugin to your project files, `sap.Logger.upload()` is called with a logon success event (for example, when the application is launched or resumed and logon is successful) so the log file is uploaded automatically. If you do not use the Settings plugin, you can upload log files only by calling the `sap.Logger.upload()` method manually.

You can build in support for logging so that an administrator can remotely set the appropriate log level from SAP Mobile Platform Server. The Kapsel Logger plugin can define each log message with specific levels, such as Debug and Error, which enables you to filter the log message by priority level. The Kapsel Logger plugin mirrors the OData logger library so that it can collect all of the logging data produced by the OData library. The Kapsel plugins use OData libraries in several places so that it can help see and trace the plugins' logging data.

Using the provided `sap.Logger.upload()` method allows you to log events that occur on the device and send them to SAP Mobile Platform Server, where an Administrator can view

them and remotely set the appropriate log level to control the amount of information that is written to the log.

This shows the index.html file for a sample app, which has the appID of "com.mycompany.logger" with the server connection information. This information allows the app to register with the appID on SAP Mobile Platform Server. This sample app logs messages with the log level and uploads a log file to SAP Mobile Platform Server. For example, to log messages with DEBUG log level, you can call the `sap.Logger.debug(...)` method. You can also use other methods for logging with other log levels (INFO, WARN and ERROR).

```
<html>
  <head>
    <script type="text/javascript" charset="utf-8"
src="cordova.js"></script>
    <script>
      logonView = null;
      logon = null;
      applicationContext = null;
      function init() {
        var appId = "com.mycompany.logger"; // Change this to
app id on server
        // Optional initial connection context var context = {
        "serverHost": "server.sap.corp", //Place your SAP
Mobile Platform server 3.0 name here
        "https": "false",
        "serverPort": "8080",
        "user": "user", //Place your user name for the OData
Endpoint here
        "password": "xxxxxxx", //Place your password for the
OData Endpoint here
        "communicatorId": "REST",
        "passcode": "password",
        "unlockPasscode": "password"
        };
        sap.Logon.init(function() { }, function() {alert("Logon
Failed"); }, appId, context, sap.logon.IabUi);
        sap.Logger.setLogLevel(sap.Logger.DEBUG);
      }

      function logMessage() {
        var employee = {name: "Dan", location : "Waterloo"};
        console.log("The value of employee is " +
JSON.stringify(employee));
      }

      function logMessage2() {
        sap.Logger.debug("Debug log message");
        sap.Logger.info("Info log message");
        sap.Logger.warn("Warn log message");
        sap.Logger.error("Error log message");
      }

      function uploadLog() {
        sap.Logger.upload(function() {
          alert("Upload Successful");
        }, function(e) {
```

```
        alert("Upload Failed. Status: " + e.statusCode + ",
Message: " + e.statusMessage);
    });
}
    document.addEventListener("deviceready", init, false);
</script>
</head>
<body>
    <h1>Logger Sample</h1>
    <button id="log" onclick="logMessage()">Log Message with
console</button><br>
    <button id="log" onclick="logMessage2()">Log Message with
Logging Plugin</button><br>
    <button id="upload" onclick="uploadLog()">Upload Log</
button>
    </body>
</html>
```

Setting the Log Level

You can manually set the Kapsel log level in the `init (...)` function by adding code, for example:

```
sap.Logger.setLogLevel(sap.Logger.INFO,
    function(logLevel) {console.log("Log level set");},
    function() {console.log("Failed to set log level");});
```

Log levels are:

- ERROR
- WARN
- INFO
- DEBUG

By default, only error level logs are captured. Use the `setLogLevel` to capture other levels. If the log level is `DEBUG`, all log level messages are stored. If it is `WARN`, the uploaded log contains `WARN` and `ERROR` messages.

On iOS, if the log level is `ERROR`, then only `ERROR` level messages are displayed in the console, even if other log level messages are generated. But if the current log level is `DEBUG`, `INFO`, or `WARN`, all generated log messages, regardless of log level, are displayed in the console. On Android, all generated log messages, regardless of log level, are shown in the Android log cat view (console).

To upload the log to the server, in the `logMessageInfoToSMP (...)` function, enter:

```
sap.Logger.setLogLevel(sap.Logger.INFO,
    function(logLevel) {console.log("Log level set");},
    function() {console.log("Failed to set log level");});
```

Limitations

On Android, the maximum for log entries is 10,000. The oldest 200 log entries are removed if the 10,000 maximum is reached. This applies to both the device and emulator.

On iOS simulators, the Logger plugin may behave in unpredictable ways, as it is intended for use with a device. On iOS devices, there is no explicit maximum for log entries, however, old messages are removed from the device after a time.

Adding the Logger Plugin

Install the Logger plugin using the Cordova command line interface.

Prerequisites

- Set up the development environment.
- Create your Cordova Project.
- Add your OS platforms.

Task

1. Add the Logger plugin by entering the following at the command prompt, or terminal:

On Windows:

```
cordova -d plugin add <SDK_HOME>\MobileSDK3\KapselSDK
\plugins\logger
```

On Mac:

```
cordova -d plugin add ~<SDK_HOME>/MobileSDK3/KapselSDK/
plugins/logger
```

Note: The path you enter to the Kapsel plugin must be the absolute path (not relative path).

2. (Optional) To see a list of installed plugins in your Cordova project, open a command prompt or terminal window, navigate to your Cordova project folder, and enter:

```
cordova plugins
```

The Cordova command line interface returns a JSON array showing installed plugins, for example:

```
[ 'org.apache.cordova.core.camera',
  'org.apache.cordova.core.device-motion',
  'org.apache.cordova.core.file' ]
```

In this example, the Cordova project has the Cordova core Camera, Accelerator (device-motion), and File plugins installed.

3. Modify the files in the www folder for the project as necessary, then copy them to the platform directories by running:

```
cordova -d prepare android
cordova -d prepare ios
```

4. Use the Android IDE or Xcode to deploy and run the project.

Note: If you are using an iOS device, remember to add the "clienthubEntitlements" to the Keychain Groups in the Entitlement section in Xcode.

Viewing Client Logs

(Applies only to hybrid) Download and view a client log associated with the selected application registration. The developer must have implemented Logger code in the application code, and the application must be registered and collecting data. The log content varies by device type and operating system.

1. From Management Cockpit, select **Registrations** on the Home screen to view application connections. Alternatively, in the **Applications** tab, click the **Registrations** tab. Information for up to 200 registered applications appears.
2. Use the search, sorting, and filtering options to locate the registration in which you are interested:
3. Click the red client log icon to display the Client Logging dialog. In some cases, a list of client logs appears.
 - a) Click **Enable Log Upload**.
 - b) Select the log level in **Log Type**.
 - c) Click **Save** to save the modified setting of whether to allow uploading of client logs, and the level at which the client should log. The red client log icon turns green.
 - d) Click a log file name to download the log and open it in your selected viewer.

Client Logs

(Applies only to hybrid) If client logging has been enabled for a hybrid application and log data is available, you can view the client long for the selected application registration.

Note: The log format varies by device type and operating system. Following are example log excerpts for Android and iOS.

Hybrid Client Log - Android Example

```
1377125811306
Debug Tag
Debug Message
null (com.sap.mp.cordova.plugins.logger.Logger:execute:54)
1893

1377125813056
Info Tag
Info Message
null (com.sap.mp.cordova.plugins.logger.Logger:execute:61)
1893

1377125814165
Warn Tag
Warn Message
null (com.sap.mp.cordova.plugins.logger.Logger:execute:68)
```

```

1893
1377125815157
Error Tag
Error Message
null (com.sap.mp.cordova.plugins.logger.Logger:execute:75)
1893

```

Hybrid Client Log - iOS Example

```

ASLMessageID = 142697
Time = Aug 20, 2013, 4:37:22 PM
TimeNanoSec = 274834000
Level = 3
PID = 4823
UID = 966313393
GID = 1824234391
ReadGID = 80
Host = PALM00545086A
Sender = KAPSEL326
Facility = com.sap.sdmlogger
Message = MAFLogon MCIM is available: NO -[MAFMCIMManager
isAvailable] Line:48 thread:<NSThread: 0x9d57c10>{name = (null), num
= 1}

ASLMessageID = 142696
Time = Aug 20, 2013, 4:37:22 PM
TimeNanoSec = 234589000
Level = 4
PID = 4823
UID = 966313393
GID = 1824234391
ReadUID = 966313393
Host = PALM00545086A
Sender = KAPSEL326
Facility = com.sap.kapsel326
Message = Finished load of: file:///Users/i834381/Library/
Application%20Support/iPhone%20Simulator/6.1/Applications/9EC4E7F3-
C156-476F-850B-56EE001EEAB2/KAPSEL326.app/www/index.html
CFLog Local Time = 2013-08-20 16:37:22.234
CFLog Thread = c07

ASLMessageID = 142695
Time = Aug 20, 2013, 4:37:22 PM
TimeNanoSec = 194786000
Level = 4
PID = 4823
UID = 966313393
GID = 1824234391
ReadUID = 966313393
Host = PALM00545086A
Sender = KAPSEL326
Facility = com.sap.kapsel326
Message = Resetting plugins due to page load.
CFLog Local Time = 2013-08-20 16:37:22.194

```

```
CFLog Thread = c07
ASLMessageID = 142694
Time = Aug 20, 2013, 4:37:22 PM
TimeNanoSec = 152145000
Level = 4
PID = 4823
UID = 966313393
GID = 1824234391
ReadUID = 966313393
Host = PALM00545086A
Sender = KAPSEL326
Facility = com.sap.kapsel326
Message = Multi-tasking -> Device: YES, App: YES
CFLog Local Time = 2013-08-20 16:37:22.151
CFLog Thread = c07
```

Testing Logging

The log file is located in `SMP_HOME\Server\log\clientlogs`
`\<application_id>\<application_registration_id>\Log.txt`.

1. Run your project with the Android IDE or Xcode.
2. In Management Cockpit, enable the upload log function.

Note: For the call to `sap.Logger.upload()` to succeed, the **Log Upload** checkbox on the registration ID in the Management Cockpit must be checked.

3. View the uploaded logs in the Management Cockpit.

Kapsel Logger API Reference

The Kapsel Logger API Reference provides usage information for Logger API classes and methods, as well as provides sample source code.

Logger namespace

The Kapsel Logger plugin provides a Cordova plugin wrapper around the SAP Mobile Platform client logging API.

It has ERROR, WARN, INFO and DEBUG log level and log messages are captured based on the selected log level. Android and iOS logger default log level is ERROR. By default only ERROR level logs are captured. The `sap.Logger.setLogLevel()` method is used to set other levels. To get log messages for all log levels, you must set the log level to DEBUG. (DEBUG < INFO < WARN < ERROR)

Using the provided `sap.Logger.upload()` method allows you to upload a log file to SAP Mobile Platform Server, where an Administrator can view them and remotely set the appropriate log

level to control the amount of information that is written to the log. When the `sap.Logger.upload()` method is triggered, a log file (not multiple) is uploaded. If the **Log Upload** check box is enabled in the Management Cockpit, the client can upload a log file by calling `sap.Logger.upload()`. If the **Log Upload** check box is disabled on the server, the client cannot upload without getting an HTTP/1.1 403 Forbidden error. For the Logger plugin, you must call `sap.Logger.upload()` to upload the log file and implement an upload button or something that does the upload from within the app to the server.

To upload a log file with the Logger plugin, these conditions must be met: 1) Log Upload check box is enabled on the server 2)The `sap.Logger.upload()` method is called by developer.

This is the expected work flow with the current architecture.

- 1) If a user has an issue, such as an exception error, he/she reports it by email or a call to customer center.
- 2) The Administrator (or developer) enables the app on the server. Then the Administrator lets the user know that he/she can upload the log file.
- 3) The user uploads a log file to the server. The Administrator gets the uploaded log file in the Management Cockpit.
- 4) The Administrator sends the file to the app's developer to debug.

On iOS, if the current log level is ERROR(default level), only ERROR level messages are displayed in the console even if other log level messages are generated. But if the current log level is DEBUG, INFO or WARN, all generated log messages, regardless of log level are displayed in the console.

On Android, all generated log messages, regardless of log level, are displayed in the Android log cat view (console).

When you use the Settings plugin, it 1) Gets the log level from the server, 2) Sets it into Logger on the client and 3) Calls `sap.Logger.upload()` upon a logon success event (when the app is launched or resumed and logon is successful). This means that the setting plugin retrieves the

selected log level(type) from the Management Cockpit on the server, sets the log level to the Logger plugin and then automatically uploads a log file to the server. If the Settings plugin is not added in the project, a log file can be uploaded only by calling the `sap.Logger.upload()` method manually by developer. In the Management Cockpit (client logging dialog box), the Log Upload check box can enable and disable and you can choose the log type(level). The uploaded log files are listed in the Management Cockpit. There are seven log types (NONE, FATAL, ERROR, WARNING, INFO, DEBUG and PATH) on the server. Since the Kapsel Logger plugin supports only DEBUG, INFO, WARN and ERROR, the Logger plugin implicitly matches FATAL to ERROR and PATH to DEBUG. If NONE is set on server admin UI, logger sets it to default log level.

Adding and Removing the Logger Plugin

The Logger plugin is added and removed using the *Cordova CLI*.

To add the Logger plugin to your project, use the following command:

```
Cordova plugin add <path to directory containing Kapsel plugins>\logger
```

To remove the Logger plugin from your project, use the following command:

```
cordova plugin rm com.sap.mp.cordova.plugins.logger
```

Members

Name	Description
<i>Logger#DEBUG</i> on page 187	Constant variable for Debug log level.
<i>Logger#ERROR</i> on page 187	Constant variable for Error log level.
<i>Logger#INFO</i> on page 188	Constant variable for Information log level.
<i>Logger#WARN</i> on page 188	Constant variable for Warning log level.

Methods

Name	Description
<i>debug(message, [tag], [successCallback], [error-Callback])</i> on page 188	Add a debug message to the log. This function logs messages with 'DEBUG' log level.
<i>error(message, [tag], [successCallback], [error-Callback])</i> on page 189	Add an error message to the log. This function logs messages with 'ERROR' log level.
<i>getLogLevel(successCallback, [errorCallback])</i> on page 191	Get log level.
<i>info(message, [tag], [successCallback], [error-Callback])</i> on page 192	Add an info message to the log. This function logs messages with 'INFO' log level.
<i>setLogLevel(level, [successCallback], [error-Callback])</i> on page 193	Set log level. This function sets the log level for logging.
<i>upload(successCallback, errorCallback)</i> on page 195	Upload a log file that contains log entries to SAP Mobile Platform server.
<i>warn(message, [tag], [successCallback], [error-Callback])</i> on page 196	Add a warning message to the log. This function logs messages with 'WARN' log level.

Source

logger.js, line 61 on page 200.

Logger#DEBUG member

Constant variable for Debug log level.

It contains "DEBUG" string.

Syntax

```
<static, constant> Logger#DEBUG : String
```

Example

```
sap.Logger.setLogLevel( sap.Logger.DEBUG );
```

Source

logger.js, line 331 on page 210.

Logger#ERROR member

Constant variable for Error log level.

It contains "ERROR" string.

Syntax

<static, constant> `Logger#ERROR` : `String`

Example

```
sap.Logger.setLogLevel(sap.Logger.ERROR);
```

Source

logger.js, line 301 on page 209.

Logger#INFO member

Constant variable for Information log level.

It contains "INFO" string.

Syntax

<static, constant> `Logger#INFO` : `String`

Example

```
sap.Logger.setLogLevel(sap.Logger.INFO);
```

Source

logger.js, line 321 on page 210.

Logger#WARN member

Constant variable for Warning log level.

It contains "WARN" string.

Syntax

<static, constant> `Logger#WARN` : `String`

Example

```
sap.Logger.setLogLevel(sap.Logger.WARN);
```

Source

logger.js, line 311 on page 210.

debug(message, [tag], [successCallback], [errorCallback]) method

Add a debug message to the log. This function logs messages with 'DEBUG' log level.

Syntax

<static> `debug(message, [tag], [successCallback], [errorCallback])`

Parameters

Name	Type	Argument	Description
<i>message</i>	String		log message to be logged
<i>tag</i>	String	(optional)	Tag can indicate what this message is for.(ex. SMP_LOGGER, SMP_AUTHPROXY) It can be appended to messages at front.
<i>successCallback</i>	function	(optional)	success callback method upon success state.When a debug message is successfully logged, it is fired. No object will be passed to success callback.
<i>errorCallback</i>	function	(optional)	error callback method upon error state. For this method, since the Kapsel Logger native code always calls success callback after executing the API of the oData logger library, it is very unusual to fire an error callback.(If Cordova caused a system exception, the error callback could be fired by Cordova.)

Example

```
sap.Logger.debug("debug message", "DEBUG_TAG");
```

Source

logger.js, line 75 on page 201.

error(message, [tag], [successCallback], [errorCallback]) method

Add an error message to the log. This function logs messages with 'ERROR' log level.

Syntax

```
<static> error( message, [tag], [successCallback], [errorCallback] )
```

Parameters

Name	Type	Argument	Description
<i>message</i>	String		log message to be logged
<i>tag</i>	String	(optional)	Tag can indicate what this message is for.(ex. SMP_LOGGER, SMP_AUTHPROXY) It can be appended to messages at front.
<i>successCallback</i>	function	(optional)	success callback method upon success state. When a debug message is successfully logged, it is fired. No object will be passed to success callback.
<i>errorCallback</i>	function	(optional)	error callback method upon error state. For this method, since the Kapsel Logger native code always calls success callback after executing the API of the oData logger library, it is very unusual to fire error callback.(If Cordova caused a system exception, the error callback could be fired by Cordova.)

Example

```
sap.Logger.error("error message", "ERROR_TAG");
```

Source

logger.js, line 150 on page 203.

getLogLevel(successCallback, [errorCallback]) method

Get log level.

This function gets current log level. Using this function, you can find out what kind of log level messages can be generated and affected at the current log level.

Syntax

```
<static> getLogLevel( successCallback, [errorCallback] )
```

Parameters

Name	Type	Argument	Description
<i>successCallback</i>	function		success callback method upon success state. When current log level is successfully retrieved, it is fired with the current log level. [DEBUG, INFO, WARN, ERROR] Log level of String type will be passed to success callback. Default log level is ERROR.
<i>errorCallback</i>	function	(optional)	error callback method upon error state. For this method, error callback is optional. Since logger native code always passes log level to success callback, it is very unusual to fire error callback. (If cordova caused system exception, error callback could be fired by cordova)

Example

```
sap.Logger.getLogLevel(function(logLevel) {
    alert("Log level is " + logLevel);
}, function() {
```

```
    alert("Failed to get log level");
  });
```

Source

logger.js, line 224 on page 206.

info(message, [tag], [successCallback], [errorCallback]) method

Add an info message to the log. This function logs messages with 'INFO' log level.

Syntax

```
<static> info( message, [tag], [successCallback], [errorCallback] )
```

Parameters

Name	Type	Argument	Description
<i>message</i>	String		log message to be logged
<i>tag</i>	String	(optional)	Tag can indicate what this message is for.(ex. SMP_LOGGER, SMP_AUTHPROXY) It can be appended to messages at front.
<i>successCallback</i>	function	(optional)	success callback method upon success state.When a debug message is successfully logged, it is fired. No object will be passed to success callback.

<i>errorCallback</i>	function	(optional)	error callback method upon error state. For this method, since the Kapsel Logger native code always calls the success callback after executing the API of the oData logger library, it is very unusual to fire an error callback. (If Cordova caused a system exception, the error callback could be fired by Cordova.)
----------------------	----------	------------	---

Example

```
sap.Logger.info("info message", "INFO_TAG");
```

Source

logger.js, line 100 on page 202.

setLogLevel(level, [successCallback], [errorCallback]) method

Set log level. This function sets the log level for logging.

Coverage of logging data in each log level: DEBUG < INFO < WARN < ERROR.

Following is the expected behavior to cover log messages at specific log levels.

Error : only error

Warn : error, warn

Info : error, warn, info

Debug : error, warn, info, debug

For example, if you want to get all log messages, you need to set it to 'Debug' level. If WARN level is set, logging data contains WARN and ERROR messages.

Default log level is ERROR.

Syntax

```
<static> setLogLevel( level, [successCallback], [errorCallback] )
```

Parameters

Name	Type	Argument	Description
<i>level</i>	String		log level to set [DEBUG, INFO, WARN, ERROR]
<i>successCallback</i>	function	(optional)	success callback method upon success state. When log level is successfully set, it is fired. No object will be passed to success callback.
<i>errorCallback</i>	function	(optional)	error callback method upon error state. For this method, since kapsel logger native code always calls success callback after executing the API of oData logger library, it is very unusual to fire error callback. (If cordova caused system exception, error callback could be fired by cordova)

Example

```
sap.Logger.setLogLevel(sap.Logger.DEBUG, function(logLevel) {
    alert("Log level set");
}, function() {
    alert("Failed to set log level");
});
```

Source

logger.js, line 175 on page 204.

upload(successCallback, errorCallback) method

Upload a log file that contains log entries to SAP Mobile Platform server.

This function uploads a log file, which is helpful for developers who want to collect logging data from the app to trace bugs and issues. It uploads a log file which contains log entries based on log level. Developers can access the log data in the Management Cockpit or a specific folder in the installed server directly.

On iOS logger, when uploading a log file, the uploaded log messages are filtered by the log level at upload. For example, when you upload a log file at the ERROR log level, uploaded log messages contain only error log level messages. When you upload a log file at the INFO level, uploaded log messages contain error, warn, and info log level messages.

Android logger just filters the generated log messages "at the log level." In other words, the already generated and filtered log messages at another log level are not affected by current log level. Log messages are not filtered at uploading. For example, set the log level to DEBUG and log four levels (DEBUG, INFO, WARN and ERROR) of messages. At this time, the Android logger has four log level messages. If you set the log level to WARN and upload a log file, the log file has four log level messages which were already generated at the DEBUG level.

Syntax

```
<static> upload( successCallback, errorCallback )
```

Parameters

Name	Type	Description
<i>successCallback</i>	function	success callback method upon success state. This method is fired when the log file is successfully uploaded (with http statusCode and statusMessage for success).

<i>errorCallback</i>	function	error callback method upon error state.If there is a connectivity error, such as an HTTP error or unknown server error, this method is fired with http statusCode and statusMessage for error.
----------------------	----------	--

Example

```
sap.Logger.upload(function() {
    alert("Upload Successful");
}, function(e) {
    alert("Upload Failed. Status: " + e.statusCode + ", Message: " + e.statusMessage);
});
```

Source

logger.js, line 250 on page 207.

warn(message, [tag], [successCallback], [errorCallback]) method

Add a warning message to the log. This function logs messages with 'WARN' log level.

Syntax

<static> warn(message, [tag], [successCallback], [errorCallback])

Parameters

Name	Type	Argument	Description
<i>message</i>	String		log message to be logged
<i>tag</i>	String	(optional)	Tag can indicate what this message is for.(ex. SMP_LOGGER, SMP_AUTHPROXY) It can be appended to messages at front.

<i>successCallback</i>	function	(optional)	success callback method upon success state. When a debug message is successfully logged, it is fired. No object will be passed to success callback.
<i>errorCallback</i>	function	(optional)	error callback method upon error state. For this method, since the Kapsel Logger native code always calls success callback after executing the API of the oData logger library, it is very unusual to fire an error callback. (If Cordova caused a system exception, the error callback could be fired by Cordova.)

Example

```
sap.Logger.warn("warn message", "WARN_TAG");
```

Source

logger.js, line 125 on page 203.

Source code***logger.js***

```
1 // ${project.version}
2 var exec = require('cordova/exec');
3
4 /**
5  * The Kapsel Logger plugin provides a Cordova plugin wrapper
6  * around the SAP Mobile Platform client logging API.
7  * <br><br>
8  *
9  *
10 *
```

Kapsel Development

8 * The Logger plugin has ERROR, WARN, INFO, and DEBUG log levels and log messages are captured based on the configured and selected log level.

9 * A Kapsel application can be set to these log levels by programmatic control, and by the administrator changing a setting on the server.

10 * For Android and iOS, the default log level is ERROR, so by default only ERROR level logs are captured.

11 * sap.Logger.setLogLevel() method is used to set other levels. If you want to get log messages at all log levels,

12 * you must set the log level to DEBUG. (DEBUG < INFO < WARN < ERROR)

13 * If the log level is set to DEBUG, the application captures all log messages.

14 * If you set the log level to INFO, the application captures INFO, WARN, and ERROR log messages.

15 * If you set the log level to WARN, the application captures WARN and ERROR log messages.

16 * If you set the log level to ERROR, the application captures only Error log messages.

17 *

18 *

19 * Using the provided sap.Logger.upload() method allows developers to upload a log file to SAP Mobile Platform Server,

20 * where an administrator can view them and remotely set the appropriate log level to control the amount of information

21 * that is written to the log. When the sap.Logger.upload() method is triggered, a log file will be uploaded.

22 * If the Log Upload checkbox is selected in the Management Cockpit, the client can upload a log file by calling sap.Logger.upload().

23 * If the Log Upload checkbox is disabled in the Management Cockpit, the client does not upload the log file to the server. The attempt to upload causes an "HTTP/1.1 403 Forbidden" error.

24 * To support manual uploading of the log, you should implement a button or some other mechanism that calls sap.Logger.upload() when needed.

25 *

26 * For the Logger plugin to upload a log file these conditions must be met: 1) Log Upload checkbox enabled in the Management Cockpit 2) sap.Logger.upload() is called by developer.

```
27      * <br>
28      * The expected work flow, with the current architecture
consists of the following: <br>
29      * 1) If a user has an issue that needs to be analyzed by an
administrator or developer, the user reports the issue as
appropriate.<br>
30      * 2) The administrator, or developer, enables the log
collection for the user on the SAP Mobile Platform server.<br>
31      * 3) The administrator lets the user know that he, or she,
can upload log file. <br>
32      * 4) The user uploads thelog file to the server, and the
administrator gets the uploaded log file in the Management
Cockpit.<br>
33      * 5) The administrator sends the file to the developer to
debug.
34      * <br><br>
35      *
36      * Currently, on iOS, if the current log level is ERROR
(default level), only ERROR level messages are displayed on the
console
37      * even if other log level messages are generated. But if the
current log level is DEBUG, INFO, or WARN,
38      * all generated log messages, regardless of log level, are
displayed on the console. <br>
39      * On Android, all generated log messages, regardless of log
level, are displayed in the Android logcat view (console)
40      * <br><br>
41      *
42      * When the Kapsel Settings plugin is added to the project,
Settings will: 1) Get log level from the server 2) Set it into Logger
on the client
43      * 3) Call sap.Logger.upload() after a logon success event,
for example, when the app is launched or resumed and logon is
successful.
44      * The Settings plugin retrieves the selected log level(type)
from the Management Cockpit on the server,
45      * sets the log level to Logger plugin, and then automatically
uploads a log file to the server.
46      * If the Settings plugin is not added to the project, a log
file can be uploaded only by the developer calling the
sap.Logger.upload() method manually.
```

Kapsel Development

47 * In the Management Cockpit, in the Client Logging dialog box, the Log Upload checkbox is able to enable or disable log file upload, and you can choose the log type(level).

48 * You can also view a list of the uploaded log files. On the server side, there are seven log types: NONE, FATAL, ERROR, WARNING, INFO, DEBUG and PATH.

49 * Since the Kapsel Logger plugin supports only DEBUG, INFO, WARN, and ERROR, the Logger plugin implicitly matches FATAL to ERROR, and PATH to DEBUG.

50 * If NONE is set in the Management Cockpit, Logger sets it to default log level.

51 *

52 * Adding and Removing the Logger Plugin

53 * Add or remove the Logger plugin using the

54 * Cordova CLI.

55 *

56 * To add the Logger plugin to your project, use the following command:

57 * Cordova plugin add <path to directory containing Kapsel plugins>\logger

58 *

59 * To remove the Logger plugin from your project, use the following command:

60 * cordova plugin rm com.sap.mp.cordova.plugins.logger

61 *

62 *

63 * @namespace

64 * @alias Logger

65 * @memberof sap

66 */

67

68 Logger = function () {

69 /**

70 * Formating for message

71 * @private

```
72     */
73     var format = function (message) {
74         if ((message === null) || (message === undefined))
75         {
76             return "";
77         }
78         return message.toString();
79     }
80
81
82     /**
83     * Add a debug message to the log.
84     * This function logs messages with a 'DEBUG' log
85     level.
86     * @memberof sap.Logger
87     * @method debug
88     * @param {String} message Log message to log.
89     * @param {String} [tag] Tag value added to the log entry
90     used to indicate the source of the message (ex. SMP_LOGGER,
91     SMP_AUTHPROXY).
92     * @param {function} [successCallback] Callback function
93     called when the message has been successfully added to the log.
94     *
95     No object will be passed
96     to success callback.
97     * @param {function} [errorCallback] Callback function
98     called when an error occurs while adding the message to the log.
99     *
100     Since Kapsel Logger native code will always call the
101     success callback function, the
102     * errorCallback function will be executed by Cordova if an
103     error or exception occurs
104     * while making the call to the plugin.
105     * @public
106     * @memberof sap.Logger
107     * @example
```

```
99         * sap.Logger.debug("debug message", "DEBUG_TAG");
100        */
101        this.debug = function (message, tag, successCallback,
errorCallback) {
102            exec(successCallback, errorCallback, "Logging",
"logDebug", [format(message), tag]);
103        }
104
105        /**
106         * Add an info message to the log.
107         * This function logs messages with the 'INFO' log
level.
108         *
109         * @memberof sap.Logger
110         * @method info
111         * @param {String} message Log message to be logged
112         * @param {String} [tag] Tag value added to the log entry
used to indicate the source of the message (for example, SMP_LOGGER,
SMP_AUTHPROXY).
113         * @param {function} [successCallback] Callback function
called when the message has been successfully added to the log.
114         * No object will be passed to success callback.
115         * @param {function} [errorCallback] Callback function
called when an error occurs while adding the message to the log.
116         * Since Kapsel Logger native code will always call the
success callback function, the
117         * errorCallback function will be executed by Cordova if
an error or exception occurs
118         * while making the call to the plugin.
119         * @public
120         * @memberof sap.Logger
121         * @example
122         * sap.Logger.info("info message", "INFO_TAG");
123        */
124        this.info = function (message, tag, successCallback,
errorCallback) {
```

```
125         exec(successCallback, errorCallback, "Logging",
"logInfo", [format(message), tag]);
126     }
127
128     /**
129     * Add a warning message to the log.
130     * This function logs messages with the 'WARN' log
level.
131     *
132     * @memberof sap.Logger
133     * @method warn
134     * @param {String} message Log message to be logged.
135     * @param {String} [tag] Tag value added to the log entry
used to indicate the source of the message (for example, SMP_LOGGER,
SMP_AUTHPROXY).
136     * @param {function} [successCallback] Callback function
called when the message has been successfully added to the log.
137     * No object will be passed to success callback.
138     * @param {function} [errorCallback] Callback function
called when an error occurs while adding the message to the log.
139     * Since Kapsel Logger native code will always call the
success callback function, the
140     * errorCallback function will be executed by Cordova if
an error/exception occurs
141     * while making the call to the plugin.
142     * @public
143     * @memberof sap.Logger
144     * @example
145     * sap.Logger.warn("warn message", "WARN_TAG");
146     */
147     this.warn = function (message, tag, successCallback,
errorCallback) {
148         exec(successCallback, errorCallback, "Logging",
"logWarning", [format(message), tag]);
149     }
150
```

Kapsel Development

```
151      /**
152      * Add an error message to the log.
153      * This function logs messages with the 'ERROR' log
154      * level.
155      *
156      * @memberof sap.Logger
157      * @method error
158      * @param {String} message log Message to be logged.
159      * @param {String} [tag] Tag value added to the log entry
160      * used to indicate the source of the message (for example, SMP_LOGGER,
161      * SMP_AUTHPROXY).
162      * @param {function} [successCallback] Callback function
163      * called when the message has been successfully added to the log.
164      * No object will be passed to success callback.
165      * @param {function} [errorCallback] Callback function
166      * called when an error occurs while adding the message to the log.
167      * Since Kapsel Logger native code will always call the
168      * success callback function, the
169      * errorCallback function will be executed by Cordova if
170      * an error or exception occurs
171      * while making the call to the plugin.
172      * @public
173      * @memberof sap.Logger
174      * @example
175      * sap.Logger.error("error message", "ERROR_TAG");
176      */
177      this.error = function (message, tag, successCallback,
178      errorCallback) {
179          exec(successCallback, errorCallback, "Logging",
180      "logError", [format(message), tag]);
181      }
182      /**
183      * Set log level.
184      * This function sets the log level for logging. <br>
```



```

177      * Coverage of logging data in each log level:  DEBUG <
INFO < WARN < ERROR. <br>
178      * Following is the expected behavior to cover log
messages at specific log levels: <br>
179      *      ERROR : only ERROR messages <br>
180      *      WARN  : ERROR and WARN messages <br>
181      *      INFO  : ERROR, WARN and INFO <br>
182      *      DEBUG : ERROR, WARN, INFO and DEBUG <br>
183      * For example, if you want to get all log messages, you
need to set the log to the 'Debug' level.
184      * If the WARN level is set, logging data contains WARN and
ERROR messages. <br>
185      * Default log level is ERROR.
186      *
187      * @memberof sap.Logger
188      * @method setLogLevel
189      * @param {String} level Log level to set [DEBUG, INFO,
WARN, ERROR]
190      * @param {function} [successCallback] Callback function
called when the log level has been successfully set.
191      * No object will be passed to success callback.
192      * @param {function} [errorCallback]  Callback function
called when an error occurs while setting the log level.
193      * Since Kapsel Logger native code will always call the
success callback function, the
194      * errorCallback function will be executed by Cordova if
an error or exception occurs
195      * while making the call to the plugin.
196      * @memberof sap.Logger
197      * @example
198      * sap.Logger.setLogLevel(sap.Logger.DEBUG,
successCallback, errorCallback);
199      *
200      * function successCallback() {
201      *     alert("Log level set");
202      * }

```

Kapsel Development

```
203      *
204      * function errorCallback() {
205      *     alert("Failed to set log level");
206      * }
207      */
208      this.setLogLevel = function (level, successCallback,
209      errorCallback) {
210          if (level.toLowerCase() === "fatal")
211              level = "ERROR";
212          else if (level.toLowerCase() === "path")
213              level = "DEBUG";
214          else if (level.toLowerCase() === "warning")
215              level = "WARN";
216          else if (level.toLowerCase() === "debug")
217              level = "DEBUG";
218          else if (level.toLowerCase() === "info")
219              level = "INFO";
220          else if (level.toLowerCase() === "error")
221              level = "ERROR";
222
223          exec(successCallback, errorCallback, "Logging",
224          "setLogLevel", [level]);
225
226      /**
227      * Get log level.
228      * This function gets the current log level.
229      * Use this function to know what kind of log level
230      * messages can be generated and affected at the current log level.
231      *
232      * @memberofof sap.Logger
233      * @method getLogLevel
```

```

233      * @param {function} successCallback Callback function
called when the log level has been successfully retrieved.
234      * When the current log level is successfully retrieved,
it is fired with the current log level. [DEBUG, INFO, WARN, ERROR]
235      * Log level of String type will be passed to success
callback.
236      * Default log level is ERROR.
237      * @param {function} [errorCallback] Callback function
called when an error occurs while getting the current log level. For
this method, error callback is optional.
238      * Since Kapsel Logger native code will always call the
success callback function, the
239      * errorCallback function will be executed by Cordova if
an error or exception occurs
240      * while making the call to the plugin.
241      * @memberof sap.Logger
242      * @example
243      * sap.Logger.getLogLevel(successCallback,
errorCallback);
244      *
245      * function successCallback(logLevel) {
246      *     alert("Log level is " + logLevel);
247      * }
248      *
249      * function errorCallback() {
250      *     alert("Failed to get log level");
251      * }
252      */
253      this.getLogLevel = function(successCallback,
errorCallback) {
254          exec(successCallback, errorCallback, "Logging",
"getLogLevel", []);
255      }
256
257      /**
258      * Upload a log file, with log entries, to SAP Mobile
Platform server.<br>

```

259 * This function uploads a log file, which is helpful for
collecting logging data from the app to trace bugs and issues.

260 * It uploads a log file, which contains log entries based
on log level.

261 * Developers can access the log data in the Management
Cockpit and/or a specific folder in installed server
directly.

262 *

263 * On iOS, the uploaded log messages are filtered by the
log level at upon upload.

264 * For example, when you upload a log file with an ERROR
log level, the uploaded log messages contain only ERROR log level
messages.

265 * When you upload log files with an INFO level, uploaded
log messages contain ERROR, WARN, and INFO log level messages.

266 *

267 *

268 * On Android, generated log messages are filtered "at the
log level."

269 * In other words, the already generated and filtered log
messages at another log level are not affected by the current log
level.

270 * Log messages are not filtered upon upload. For example,
if you set the log level to DEBUG log messages are filtered at four
levels (DEBUG, INFO, WARN, and ERROR.

271 * Logger on Android has four log levels messages. So, if
you set the log level to WARN and upload a log file, the log file has
four log level messages that were already generated at the DEBUG
level.

272 *

273 * @memberof sap.Logger

274 * @method upload

275 * @param {function} successCallback Callback function
called when a log file is successfully uploaded to the server.

276 * When a log file is successfully uploaded, it is fired.
(with http statusCode and statusMessage for success)

277 * @param {function} errorCallback Callback function
called when an error occurs while uploading a log file to the
server.

278 * If there is a connectivity error, such as an HTTP error,
or unknown server error,

```
279      * it is fired with http statusCode and statusMessage for
error.
280      * @public
281      * @memberof sap.Logger
282      * @example
283      * sap.Logger.upload(successCallback, errorCallback);
284      *
285      * function successCallback() {
286      *     alert("Upload Successful");
287      * }
288      *
289      * function errorCallback(e) {
290      *     alert("Upload Failed. Status: " + e.statusCode + ",
Message: " + e.statusMessage);
291      * }
292      */
293      this.upload = function (successCallback, errorCallback)
{
294          sap.Logon.unlock(function (connectionInfo) {
295              //Add application ID required for REST call
296              connectionInfo.applicationId =
sap.Logon.applicationId;
297
298              exec(successCallback, errorCallback, "Logging",
"uploadLog", [connectionInfo]);
299          }, function () {
300              errorCallback({statusCode : 0, statusMessage :
"Logon failed"});
301          });
302      }
303  }
304
305  /**
306      * Constant variable for Error log level. It contains "ERROR"
string.
```

```
307      * @memberof sap.Logger
308      * @constant
309      * @type String
310      * @example
311      * sap.Logger.setLogLevel(sap.Logger.ERROR);
312      */
313      Logger.prototype.ERROR = "ERROR";
314
315      /**
316      * Constant variable for Warning log level. It contains "WARN"
string.
317      * @memberof sap.Logger
318      * @constant
319      * @type String
320      * @example
321      * sap.Logger.setLogLevel(sap.Logger.WARN);
322      */
323      Logger.prototype.WARN = "WARN";
324
325      /**
326      * Constant variable for Information log level. It contains
"INFO" string.
327      * @memberof sap.Logger
328      * @constant
329      * @type String
330      * @example
331      * sap.Logger.setLogLevel(sap.Logger.INFO);
332      */
333      Logger.prototype.INFO = "INFO";
334
335      /**
336      * Constant variable for Debug log level. It contains "DEBUG"
string.
```

```

337     * @memberof sap.Logger
338     * @constant
339     * @type String
340     * @example
341     * sap.Logger.setLogLevel(sap.Logger.DEBUG);
342     */
343     Logger.prototype.DEBUG = "DEBUG";
344
345     module.exports = new Logger();
346

```

Using the Kapsel Push Plugin

The Push notification plugin enables push notification capability for Kapsel applications.

Push Plugin Overview

The push notification APIs enable you to send push data to Kapsel applications.

The push notification system consists of:

1. The Kapsel application, which runs on the device and receives the notifications.
2. The notification service provider, for example, APNS for Apple devices, and GCM for Android devices.
3. The SAP Mobile Platform Server, which collects device IDs from the clients and push notifications through the notification service provider.

The Kapsel Push plugin allows you to enroll applications for notification with notification registration, as well as to receive and process incoming notifications for Kapsel applications. This plugin also supports background notification processing.

In a typical deployment, SAP Mobile Platform Server sends push messages to a push server through a RESTful API, which in turn delivers the push message to the user agent, which then provides execution instructions for the app. The user agent then delivers the push message to the designated app.

The push API tasks include:

- Registering and unregistering a push notification
- Push notification handling
- Push notification configuration
- Error message handling

Note: As a best practice, you should rarely use the unregister function. It is explained in detail at <http://developer.android.com/google/gcm/adv.html#unreg>.

Provisioning Devices for Push

You must register your device with a notification service, such as Apple Push Notification Service (APNS) for Apple devices, or Google Cloud Messaging (GCM) for Android devices.

In a production environment, when you register your application with the notification service provider, the device ID (iOS) or the registration ID (Android), is sent to the SAP Mobile Platform server. For iOS, the push certificate is stored there, and is used to authenticate push requests to the APNS server. When a push request is processed, that information is then used to target specific apps running on individual devices.

Provision the iOS Device for APNS

SAP Mobile Platform provides support for Apple Push Notification Service (APNS) by pushing notifications to Kapsel while it is offline.

With APNS, each device establishes encrypted IP connections to the service and receives notifications about availability of new items that are awaiting retrieval from the server. On 3G networks, this feature overcomes network issues with always-on connectivity and battery life consumption.

Note: APNS cannot be used on a simulator.

Examples of cases when notifications are sent include when the server identifies that a new message needs to be sent to the device, for example, when a new app is assigned to the device, or a push notification message is sent to the server and targeting a particular user when the app is not running.

See the *Apple Local and Push Notification Programming Guide at Provisioning and Development*, where APNS is documented in detail.

Register with Apple to download and use the iOS SDK to develop with the simulator. To deploy applications to devices, you must create a certificate in your developer account and provision your device.

Generating a Certificate Request File

Create a certificate signing request file to use for authenticating the creation of the SSL certificate.

1. Launch the Keychain Access application on your Mac (usually found in the **Applications > Utilities** folder).
2. Select **Keychain Access > Certificate Assistant > Keychain Access**.
3. Enter your e-mail address and name, then select **Save to disk**, and click **Continue**.
This downloads the `.certSigningRequest` file to your desktop.

Creating an App ID

Create a new App ID for the application.

As a convention, the App ID is in the form of a reversed address, for example, `com.example.MyPushApp`. The App ID must not contain a wildcard character ("*").

1. Go to the *Apple Developer Member Center* Web site, log in, if required, and select *Certificates, Identifiers & Profiles*.
2. Select **Identifiers > App IDs**, and click the +.
3. Enter a name for your App ID, and, under App Service, select **Push Notifications**.
4. Accept the default App ID prefix, or choose another one.
5. Under App ID Suffix, select **Explicit App ID**, and enter your iOS app's Bundle ID. This string should match the Bundle Identifier in your iOS app's `Info.plist`.
6. Select **Continue**.
Verify that all the values are correct. Push Notifications should be enabled, and the Identifier field should match your app's Bundle Identifier (plus App ID Prefix).
7. Click **Submit**.

Configuring the App ID for Push Notifications

Once you create an App ID, you must configure it for push notifications.

1. From the list of iOS App IDs, select the App ID to configure, then select **Settings**.
2. Scroll down to the Push Notifications section and, under Development SSL Certificate, select **Create Certificate**.
Here you can create both a Development SSL Certificate and a Production SSL Certificate.
3. Follow the instructions for creating a Certificate Signing Request (CSR), select **Continue**, then select **Choose File** to locate the `.certSigningRequest` you created.
4. Click **Generate**.
5. Click **Done** once the certificate is ready, and download the generated SSL certificate from the iOS App ID Settings screen.
6. Install the SSL in your Keychain.
 - a) In Keychain Access, under My Certificates, find the certificate you just added, right-click on it, select **Export Apple Development IOS Push Services**, and save it as a `.p12` file.

Note: Do not enter an export password when prompted. You may, however, need to enter your OSX password to allow Keychain Access to export the certificate from your keychain.

Creating the Provisioning File

Create a provisioning profile to authenticate your device to run the app you are developing.

If you create a new App ID or modify an existing one, you must regenerate and install your provisioning file.

1. Navigate to the *Apple Developer Member Center* Web site, and select *Certificates, Identifiers & Profiles*.
2. From the iOS Apps section, select **Provisioning File**, and select the + button to create a new provisioning file.
3. Choose **iOS App Development** as your provisioning profile type, then click **Continue**.
4. From the drop-down, choose the App ID you created and click **Continue**.
5. Select your iOS Development certificate in the next screen, and click **Continue**.
6. Select which devices to include in the provisioning profile, and click **Continue**.
7. Choose a name for your provisioning profile, then click **Generate**.
8. Click **Download** to download the generated provisioning file.
9. Double-click the downloaded provisioning file to install it.

Xcode's Organizer opens in the Devices pane. Your new provisioning profile appears in the Provisioning Profiles section of your Library. Verify that the status for the profile is "Valid profile." If the profile is invalid, verify that your developer certificate is installed in your Keychain.

Provision Android Devices for Push

Use this procedure to provision your Android device for Google Cloud Messaging Service (GCM).

Configuring Google Cloud Messaging Service

Google Cloud Messaging (GCM) is a service that allows you to send data from the server to Android devices, and also to receive messages from devices on the same connection.

For information about GCM, see <http://developer.android.com/google/gcm/gs.html>.

1. Open the Google APIs Console page and select **Services**.
2. Turn the Google Cloud Messaging toggle to **ON**.
3. Accept the terms of the Google APIs Terms of Service.
4. Create a new server key.
 - a) Select **API Access**.
 - b) Click **Create new Server key**.

Either a server key or browser key work. If you use a server key, it allows you to whitelist IP addresses.

Note: If you get a "Sender Mismatch ID" error when using a Server key, use the Browser key instead. You can find your ProjectID from the URL in your Google API

console. Usually, it looks similar to this: <https://code.google.com/apis/console/#project:348986612458>, where 348986612458 is the ProjectID.

- c) Click **Create**.

Take note of the API key value, as you need this to register.

Adding the Push Notification Plugin

Install the Push plugin using the Cordova command line interface.

Prerequisites

- Set up the development environment.
- Create your Cordova Project.
- Add your OS platforms.

Task

When you add the Push plugin to your project, the Settings and Logger plugins are also added automatically.

1. Add the Push plugin by entering the following at the command prompt, or terminal:

On Windows:

```
cordova -d plugin add <SDK_HOME>\MobileSDK3\KapselSDK
\plugins\push
```

On Mac:

```
cordova -d plugin add ~<SDK_HOME>/MobileSDK3/KapselSDK/
plugins/push
```

Note: The path you enter to the Kapsel plugin must be the absolute path (not relative path).

2. (Optional) To see a list of installed plugins in your Cordova project, open a command prompt or terminal window, navigate to your Cordova project folder, and enter:

```
cordova plugins
```

The Cordova command line interface returns a JSON array showing installed plugins, for example:

```
[ 'org.apache.cordova.core.camera',
  'org.apache.cordova.core.device-motion',
  'org.apache.cordova.core.file' ]
```

In this example, the Cordova project has the Cordova core Camera, Accelerator (device-motion), and File plugins installed.

3. Modify the files in the `www` folder for the project as necessary, then copy them to the platform directories by running:

```
cordova -d prepare android
cordova -d prepare ios
```

4. Use the Android IDE or Xcode to deploy and run the project.

Note: If you are using an iOS device, remember to add the "clienthubEntitlements" to the Keychain Groups in the Entitlement section in Xcode.

Configuring Push on SAP Mobile Platform Server

You must explicitly register the application connection using the Management Cockpit.

1. Start the Management Cockpit.
2. Select **Applications**, and click **New**.
3. In the **New Application** window, enter values.

Field	Value
ID	Unique identifier for the application in reverse domain notation.
Name	Application name.
Vendor	(Optional) Vendor who developed the application.
Version	Application version. Currently, only version 1.0 is supported.
Type	Application type. <ul style="list-style-type: none"> • Native – native iOS and Android applications. • Hybrid – container-based applications, such as Kapsel. • Agency – metadata-driven applications, such as Agency. Application configuration options differ depending on your selection.
Description	(Optional) Short description of the application.

4. Click the **Backend** tab and configure the endpoint information.
5. Click the **Push** tab to configure the push settings.
 - For Android GCM, see *Android Push Notifications*.
 - For Apple APNS, see *Apple Push Notifications*
6. In the settings for your device, enable the app to receive push notifications.

Android Push Notifications

Configure Android push notifications for the selected application, to enable client applications to receive Google Cloud Messaging (GCM) notifications.

1. From Management Cockpit, select **Applications > Push**.
2. Under Android, enter the access key for API key. This is the access key you obtained for your Google API project (<http://developer.android.com/google/gcm/gs.html>).
3. Enter a value for Sender ID. This is the project identifier.

4. (Optional) Configure push notifications for each device type supported.

Apple Push Notifications

Configure Apple Push Notifications for the selected application, to enable client applications to receive APNS notifications.

1. From Management Cockpit, select **Applications > Push**.
2. Under Apple, select **APNS endpoint**. "None" is the default endpoint value for all the applications.
3. Select **Sandbox** to configure APNS in a development and testing environment, or **Production** to configure APNS in a production environment.
 - a) Click **Browse** to navigate to the certificate file.
 - b) Select the file, and click **Open**.
 - c) Enter a valid password.

Note: The default URL is for a production environment; for a development and testing environment, change the URL to gateway.sandbox.push.apple.com.

4. (Optional) Configure push notifications for each device type supported.

Testing Push Notifications

Test the push and settings plugins.

1. Open the project in your development IDE.
2. Build and run the project.
3. Send a REST request to send a notification to the Kapsel app.

Sample Application for Android

You can use this code to test the Push and Settings APIs on Android.

Make sure you examine the code carefully and make the necessary changes as explained in the comments.

```
<html>
  <head>
    <script src="cordova.js"></script>
    <script>
      applicationContext = null;
      appId = "bobapp2"; //Place your application id here

      smpURL = null;

      function init() {
        // Optional initial connection context
        var context = {
          "serverHost": "machine_name.com", //Place your SMP
          "https": "false",
          "server name here
3.0
```

```

        "serverPort": "80",
        "user": "smpAdmin", //Place your user name for the
OData Endpoint here
        "password": "s3pAdmin", //Place your password for
the OData Endpoint here
        "communicatorId": "REST",
        "passcode": "password",
        "unlockPasscode": "password"
    };
    sap.Logon.init(logonSuccessCallback, function()
{ alert("Logon Failed"); }, appId, context, sap.Logon.IabUi);
    sap.Logger.setLogLevel(sap.Logger.DEBUG);
}

function register() {
    try {
        sap.Logon.registerOrUnlock(logonSuccessCallback,
errorCallback);
    }
    catch (e) {
        alert("Problem with register");
    }
}

function unregister() {
    try {
        sap.Logon.core.deleteRegistration(logonUnregisterSuccessCallback,
errorCallback);
    }
    catch (e) {
        alert("problem with unregister");
    }
}

function logonSuccessCallback(result) {
    console.log("logonSuccessCallback " +
JSON.stringify(result));
    if (result) { //calling registerOrUnlock returns null
the second time it is called. Possible bug.
        applicationContext = result;
        smpURL = applicationContext.applicationEndpointURL;
        console.log(smpURL);
        if (smpURL.charAt(smpURL.length - 1) == "/") {
            smpURL = smpURL.substring(0,
applicationContext.applicationEndpointURL.length - 1);
        }
        console.log(smpURL);
        smpURL = smpURL.substring(0,
smpURL.lastIndexOf("/"));
        console.log(smpURL);
    }
}

function logonUnregisterSuccessCallback(result) {
    console.log("logonUnregisterSuccessCallback " +

```

```

JSON.stringify(result));
    applicationContext = null;
}

function errorCallback(e) {
    alert("An error occurred");
    alert(JSON.stringify(e));
}

function registerForPush() {
    var nTypes = sap.Push.notificationType.SOUNDS |
sap.Push.notificationType.ALERT | sap.Push.notificationType.BADGE;
    sap.Push.registerForNotificationTypes(nTypes,
regSuccess, regFailure, processNotification, "186452565698"); //
GCM Sender ID, null for APNS

}

function unregisterForPush() {
    var nTypes = sap.Push.notificationType.SOUNDS |
sap.Push.notificationType.ALERT;
sap.Push.unregisterForNotificationTypes(unregCallback);
}

function regSuccess(msg) {
    alert("Successfully registered"+msg);
}

function regFailure(errorInfo) {
    alert("Failed to register");
    alert(JSON.stringify(errorInfo));
    console.log("Error while registering. " +
JSON.stringify(errorInfo));
}

function unregCallback(msg) {
    alert("In unregCallback with params");
    console.log("Unregistered" + JSON.stringify(msg));
}

function unregCallback() {
    alert("In unregCallback with no params");
}

function processNotification(notification) {
    console.log("Received a notification: " +
JSON.stringify(notification));
}

function processMissedNotification(notification) {
    alert("In processMissedNotification");
    console.log("In processMissedNotification");
    console.log("Received a missed notification: " +
JSON.stringify(notification));
}

```

```

        function checkForNotification(notification) {
sap.Push.checkForNotification(processMissedNotification);
        }

        function showRegistrationInfo() {
            xmlhttp = new XMLHttpRequest();
            var url = smpURL + "/odata/applications/latest/" +
appId + "/Connections('" +
applicationContext.applicationConnectionId + "')";
            alert(url);
            xmlhttp.open("GET", url, false);
            xmlhttp.setRequestHeader("X-SMP-APPCID",
applicationContext.applicationConnectionId);
            xmlhttp.send();
            var responseText = xmlhttp.responseText;
            alert(responseText);
            console.log(responseText);
        }

        function getBadgeCallback(data) {
            alert("The badge number is : " + data["badgecount"]);
        }

        function getBadgeNum(){
            if(device.platform == "Android"){
                alert("badge number is iOS only!");
                return;
            }
            sap.Push.getBadgeNumber(getBadgeCallback);

        }

        function badgeCallBack(msg) {
            alert("Set badget number : " + msg);
        }

        function setBadgeNum(){
            if(device.platform == "Android"){
                alert("badge number is iOS only!");
                return;
            }

            sap.Push.setBadgeNumber(10, badgeCallBack);

        }

        function resetBadgeCallback(msg){
            alert("Reset badge number : " + msg);
        }

        function resetBadgeNum(){

            if(device.platform == "Android"){

```



```

        alert("badge number is iOS only!");
        return;
    }
    sap.Push.resetBadge(resetBadgeCallback);
}

document.addEventListener("deviceready", init, false);
</script>
</head>
<body>
    <h1>Push</h1>
    <button onclick="registerForPush()">Register For Push</
button><br><br>
    <button onclick="unregisterForPush()">Unregister For Push</
button><br><br>
    <button onclick="checkForNotification()">Check for
Notification</button><br><br>
    <button onclick="showRegistrationInfo()">Registration Info</
button><br><br>
    <button onclick="getBadgeNum()">Get Badge Number</
button><br><br>
    <button onclick="setBadgeNum()">Set Badge Number</
button><br><br>
    <button onclick="resetBadgeNum()">Reset Badge Number to 0</
button><br>
</body>
</html>

```

Notification Data Sent Through HTTP Headers

Notification data can be sent by the back end as a generic HTTP headers or as device platform-specific HTTP headers.

The notification URL is:

```
http[s]://<host:port/Notification>/<registration id>
```

Note: Applications built in SAP Mobile Platform 3.0 and later should adopt the header format X-SMP-XXX. To maintain backward compatibility, applications built in earlier versions can continue to use the header format X-SUP-XXX. However, X-SUP-XXX headers will be removed in future releases.

- **Generic header**

The generic HTTP header is used in the HTTP request to send any notification type such as APNS, GCM, Blackberry, or WNS.

Header format for notification data in SAP Mobile Platform 3.x and later:

```
<X-SMP-DATA>
```

- **APNS-specific headers**

Use these APNS-specific HTTP headers to send APNS notifications via SAP Mobile Platform:

Header Structure (SAP Mobile Platform and later)	Consists of
<X-SMP-APNS-ALERT>	A JSON document. You can use this header or other individual headers listed in this table.
<X-SMP-APNS-ALERT-BODY>	Text of the alert message.
<X-SMP-APNS-ALERT-ACTION-LOC-KEY>	If a string is specified, this header shows an alert with two buttons: Close and View . iOS uses the string as a key to get a localized string for the correct button title instead of View . If the value is null, the system shows an alert. Clicking OK dismisses the alert.
<X-SMP-APNS-ALERT-LOC-KEY>	Key to an alert-message string in a <code>Localizable.strings</code> file for the current localization.
<X-SMP-APNS-ALERT-LOC-ARGS>	Variable string values to appear in place of the format specifiers in <code>loc-key</code> .
<X-SMP-APNS-ALERT-LAUNCH-IMAGE>	File name of an image file in the application bundle. It may include the extension. Used as the launch image when you tap the action button or move the action slider. If this property is not specified, the system uses one of the following: <ul style="list-style-type: none"> • The previous snapshot • The image identified by the <code>UILaunchImage-File</code> key in the <code>Info.plist</code> file of the application • The <code>Default.png</code>.
<X-SMP-APNS-BADGE>	Number that appears as the badge on the application icon.
<X-SMP-APNS-SOUND>	Name of the sound file in the application bundle.
<X-SMP-APNS-DATA>	Custom payload data values. These values must use the JSON-structured and primitive types, such as dictionary (object), array, string, number, and Boolean.

For additional information about APNS headers, see the Apple Web site: <http://developer.apple.com/library/mac/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html>.

- **GCM-specific headers**

Use these GCM-specific HTTP headers to send GCM notifications:

Header Structure (SAP Mobile Platform and later)	Consists of
<X-SMP-GCM-COLLAPSEKEY >	An arbitrary string (such as "Updates Available") that collapses a group of like messages when the device is offline, so that only the last message is sent to the client. Note: If you do not include this header, the default value "Updates Available, is used
<X-SMP-GCM-DATA>	Payload data, expressed as parameters prefixed with data and suffixed as the key.
<X-SMP-GCM-DELAYWHILEIDLE>	(Optional) Represented as 1 or true for true, any other value for false, which is the default value.
<X-SMP-GCM-TIMETOLIVE>	Time (in seconds) that the message remains available on GCM storage if the device is offline.

For additional information about GCM headers, see the Android Web site: <http://developer.android.com/guide/google/gcm/gcm.html#send-msg>.

- **BES/BIS-specific header**

Use the BlackBerry-specific HTTP header to send BES/BIS notifications:

<x-sup-rim-data> or <X-SMP-RIM-DATA>

- **WNS specific header**

Use these HTTP headers to send Windows 8 desktop and tablet application notifications:

Header Structure (SAP Mobile Platform and later)	Consists of
<X-SMP-WNS-DATA>	Send payload data to the device as raw notification. Payload data may also be a binary data encoded as a Base64-encoded string. Size should not exceed 5KB.
<X-SMP-WNS-ALERT>	Text string of the notification, as Tile and Toast notifications.
<X-SMP-WNS-BADGE>	Number that appears as the badge on the application icon.

- **MPNS (Notification for Windows Phone)**

Use these Windows Phone-specific HTTP headers to send MPNS notifications:

Request Header Structure	Consists of
<X-SMP-MPNS-DATA>	Send payload data to device as raw notification. Payload data may also be a binary data encoded as a Base64-encoded string. String length should not exceed more than 2900 characters.
<X-SMP-MPNS-ALERT>	Text string of the notification, as Tile and Toast notifications.
<X-SMP-MPNS-BADGE>	Number that appears as the badge on the application icon.

Kapsel Push API Reference

The Kapsel Push API Reference provides usage information for Push API classes and methods, as well as provides sample source code.

Push namespace

The push plugin provides an abstraction layer over the

Google Cloud Messaging for Android (GCM) and Apple Push Notification Service (APNS).

A notification can be sent to a device registered with an application through a rest call at `http://SMP_3.0_SERVER:8080/Notifications/application_registration_id`

Adding and Removing the Push Plugin

The Push plugin is added and removed using the *Cordova CLI*.

To add the Push plugin to your project, use the following command:

```
cordova plugin add <path to directory containing Kapsel plugins>push
```

To remove the Push plugin from your project, use the following command:

```
cordova plugin rm com.sap.mp.cordova.plugins.push
```

Methods

Name	Description
<i>checkForNotification(callback)</i> on page 225	This method checks for any notifications received while the application was not running in the foreground.
<i>getBadgeNumber(callback)</i> on page 226	Used to fetch the badge count for the application.
<i>registerForNotificationTypes(types, successCallback, errorCallback, notificationlistenerfunc, [senderId])</i> on page 227	Function called by the application to register notification types to receive.
<i>resetBadge(callback)</i> on page 228	Used to reset the badge count for the application.
<i>setBadgeNumber(number, callback)</i> on page 229	Used to set the badge count for the application.
<i>unregisterForNotificationTypes(callback)</i> on page 229	Function called by the application to unregister from future notifications.

Type Definitions

Name	Description
<i>callback([devtok])</i> on page 230	This method updates the application with the new device token in the SAP Mobile Platform server.

Source

push.js, line 29 on page 232.

checkForNotification(callback) method

This method checks for any notifications received while the application was not running in the foreground.

Application developer can call this function directly or register with an event handler to be called automatically. It is ok to call this function even if the device is not yet registered for push notification.

Syntax

```
<static> checkForNotification( callback )
```

Parameters

Name	Type	Description
<i>callback</i>	function	The callback function that receives the notification. The callback function will receive a string as its argument. This string will contain the notification message sent from the server intact.

Example

```
function processBackgroundMessage (mesg) {
}
function checkBackgroundNotification() {
    sap.Push.checkForNotification (processBackgroundMessage);
}
document.addEventListener ("onSapLogonSuccess",
checkBackgroundNotification, false);
document.addEventListener ("onSapResumeSuccess",
checkBackgroundNotification, false);
```

Source

push.js, line 340 on page 244.

***getBadgeNumber(callback)* method**

Used to fetch the badge count for the application.

This function is used only by iOS. Other platforms do not have the badge count concept.

Syntax

<static> getBadgeNumber(*callback*)

Parameters

Name	Type	Description
<i>callback</i>	function	Success callback to call when to send the badge count. The callback function will contain an argument in json format with the current badge count. Look into the example for the detail on how to use them.

Example

```
function getBadgeNumCallback(data) { badgecount =
data["badgecount"]; }
sap.Push.getBadgeNumber (getBadgeNumCallback);
```

Source

push.js, line 247 on page 240.

registerForNotificationTypes(types, successCallback, errorCallback, notificationlistenerfunc, [senderId]) method

Function called by the application to register notification types to receive.

Syntax

```
<static> registerForNotificationTypes( types, successCallback, errorCallback,
notificationlistenerfunc, [senderId] )
```

Parameters

Name	Type	Argument	Description
<i>types</i>	string		Types of notifications the application wants to receive. The different types of notifications are expressed in <code>notification-Type</code> Notification types allowed are Disable all notifications (NONE: 0), Set badge count on app icon (BADGE: 1), Play sounds on receiving notification (SOUNDS: 2) and Show alert on receiving notification (ALERT: 4).
<i>successCallback</i>	string		Success callback to call when registration is successful.

<i>errorCallback</i>	string		Error callback to call when registration attempt fails.
<i>notificationlistener-func</i>	string		The function that receives the notification for processing by the application.
<i>senderId</i>	string	(optional)	The sender ID that is used for GCM registration. For other platforms it is null.

Example

```
regid = "211112269206";
function registerSuccess(msg) {}
function registerFailure(msg) {}
function ProcessNotification(msg) {}
sap.Push.registerForNotificationTypes(sap.Push.notificationType.badge | sap.Push.notificationType.sound | sap.Push.notificationType.alert, registerSuccess, registerFailure, ProcessNotification, regid);
```

Source

push.js, line 200 on page 238.

resetBadge(callback) method

Used to reset the badge count for the application.

This function is used only by iOS. Other platforms do not have the badge count concept.

Syntax

<static> resetBadge(*callback*)

Parameters

Name	Type	Description
<i>callback</i>	function	Success callback to call when the badge count is reset. The callback function will contain an argument in string format. This argument can be used for informative purpose.

Example

```
function badgeCallback (msg) { }
sap.Push.resetBadge (badgeCallback) ;
```

Source

push.js, line 284 on page 242.

setBadgeNumber(number, callback) method

Used to set the badge count for the application.

This function is used only by iOS. Other platforms do not have the badge count concept.

Syntax

```
<static> setBadgeNumber( number, callback )
```

Parameters

Name	Type	Description
<i>number</i>	number	The badge count to set for the application.
<i>callback</i>	function	Success callback to call when to send the badge count. The callback function will contain an argument in string format. This argument can be used for informative purpose.

Example

```
function badgeCallback (msg) { }
badgenum = 10;
sap.Push.setBadgeNumber (badgenum, badgeCallback) ;
```

Source

push.js, line 265 on page 241.

unregisterForNotificationTypes(callback) method

Function called by the application to unregister from future notifications.

Syntax

```
<static> unregisterForNotificationTypes( callback )
```

Parameters

Name	Type	Description
<i>callback</i>	function	Success callback to call when deregistration is successful. This callback function will contain a string with a message. This message is just for informative purpose.

Example

```
function unregCallback(msg) {}
sap.Push.unregisterForNotificationTypes(unregCallback);
```

Source

push.js, line 229 on page 240.

***callback([devtok])* type**

This method updates the application with the new device token in the SAP Mobile Platform server.

Syntax

```
<static> callback( [devtok] )
```

Parameters

Name	Type	Argument	Description
<i>devtok</i>	string	(optional)	The device token received from the APNS/GCM device registration.

Example

```
function callback(msg) {}
devToken = "123123213213"; //sample device token
sap.Push.updateWithDeviceToken(devToken, callback);
```

Source

push.js, line 305 on page 242.

Source code

push.js

```

1      // ${project.version}
2      var exec = require("cordova/exec");
3
4      /**
5       * The Kapsel Push plugin provides an abstraction layer over
the
6       * <a href="http://developer.android.com/google/gcm/
index.html">Google Cloud Messaging for Android (GCM)</a>
7       * and
8       * <a href="http://developer.apple.com/library/mac/
documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/
Chapters/ApplePushService.html#//apple_ref/doc/uid/TP40008194-
CH100-SW9">Apple Push Notification Service (APNS)</a>.
9       * <br/><br/>
10      * A notification can be sent to a device registered with an
application through a
11      * rest call at <pre>http://SMP_3.0_SERVER:8080/
Notifications/application_registration_id</pre>
12      * <br/><br/>
13      * <b>Adding and Removing the Push Plugin</b><br/>
14      * Add or remove the Push plugin using the
15      * <a href="http://cordova.apache.org/docs/en/edge/
guide_cli_index.md.html#The%20Command-line%20Interface">Cordova
CLI</a>.<br/>
16      * <br/>
17      * To add the Push plugin to your project, use the following
command:<br/>
18      * cordova plugin add <path to directory containing Kapsel
plugins>\push<br/>
19      * <br/>
20      * To remove the Push plugin from your project, use the
following command:<br/>
21      * cordova plugin rm com.sap.mp.cordova.plugins.push
22      * <br/>
23      *

```

Kapsel Development

```
24     * @namespace
25     * @alias Push
26     * @memberof sap
27     */
28
29     module.exports = {
30
31
32
33         /**
34          * Helper method for handling failure callbacks. It is
35          * configured as a failure callback in call_native()
36          *
37          * @param {msg} Error message with the cause of failure
38          *
39          * @private
40          * @name failure
41          * @function
42          */
43
44         failure: function (msg) {
45             sap.Logger.debug("Javascript Callback Error: " +
46                 msg, "PUSHJS", function(m) {}, function(m) {});
47         },
48
49         /**
50          * Helper method for calling native methods.
51          *
52          * @param {function} callback
53          * @param {string} Name of the action to invoke on the
54          * plugin
```

```

54     * @param {array} List of arguments
55     * @private
56     * @name call_native
57     * @function
58     */
59     call_native: function (callback, name, args) {
60
61         if(arguments.length == 2) {
62             args = []
63         }
64         ret = exec(
65             callback,          /** Called when signature
capture is successful */
66             sap.Push.failure,  /** Called when
signature capture encounters an error */
67             'SMPPushPlugin',   /** Tell Cordova that to
run "PushNotificationPlugin" */
68             name,              /** Tell the plugin the
action to perform */
69             args);             /** List of arguments to
the plugin */
70         return ret;
71     },
72
73     /**
74     * Helper method to check if platform is iOS.
75     *
76     * @return {bool} Whether the current platform is iOS or
not.
77     * @private
78     * @name isPlatformIOS
79     * @function
80     */
81     isPlatformIOS: function () {

```

```
82     return device.platform == "iPhone" || device.platform ==
"iPad" || device.platform == "iPod touch" || device.platform ==
"iOS"
83     },
84     /**
85     * Function called by the application to get connection
information.
86     *
87     * @param {string} [types] Types of notifications the
application wants to receive. The different types of notifications
are expressed in <code>notificationType</code>
88     * @param {string} [successCB] Success callback to call
when registration is successful.
89     * @param {string} [errorCB] Error callback to call when
registration attempt fails.
90     * @private
91     * @memberof sap.Push
92     * @function getConnectionSettings
93     * @example
94     * sap.Push.getConnectionSettings(function(){
95     *     sap.Logger.debug("getting Connection
Settings","PUSHJS",function(m){},function(m){});
96     *     console.log("getting Connection Settings");
97     *     sap.Push.registerForNotification(types,
successCallback, errorCallback, notificationListenerFunc,
senderId );
98     **/
99     getConnectionSettings : function (successCB, errorCB) {
100
101
102         if (sap.Settings.isInitialized == true)
103             {
104                 /*It is already initialized */
105                 successCB();
106
107             } else {
```

```
108         sap.Settings.isInitialized = true;
109         var pd = "";
110         sap.Logon.unlock(function (connectionInfo) {
111             var userName =
connectionInfo["registrationContext"]["user"];
112             var password =
connectionInfo["registrationContext"]["password"];
113             var applicationConnectionId =
connectionInfo["applicationConnectionId"];
114             var securityConfig =
connectionInfo["registrationContext"]["securityConfig"];
115             var endpoint =
connectionInfo["applicationEndpointURL"];
116             var keySSEnabled = "false";
117             var splitendpoint =
endpoint.split("/");
118             if (splitendpoint[0] ==
"https:")
119             {
120                 keySSEnabled="true";
121             }
122             if (securityConfig == null) {
123                 securityConfig = "";
124             }
125             var burl = splitendpoint[2];
126             var appId = splitendpoint[3];
127             pd = appId+userName+password;
128             sap.Settings.store = new
sap.EncryptedStorage("SettingsStore", pd);
129             connectionData = {
130             "keyMAFLogonOperationContextConnectionData": {
131             "keyMAFLogonConnectionDataApplicationSettings":
132             {
133             "DeviceType":device.platform,
```

Kapsel Development

```
134     "DeviceModel":device.model,
135     "ApplicationConnectionId":applicationConnectionId
136         },
137     "keyMAFLogonConnectionDataBaseURL":burl
138         },
139     "keyMAFLogonOperationContextApplicationId":appId,
140     "keyMAFLogonOperationContextBackendUserName":userName,
141     "keyMAFLogonOperationContextBackendPassword":password,
142     "keyMAFLogonOperationContextSecurityConfig":securityConfig,
143     "keySSEEnabled":keySSEEnabled
144         };
145     sap.Settings.start(connectionData,
146                                     function (msg)
147     {
148         sap.Settings.isInitialized = true;
149         sap.Logger.debug("Setting Exchange is succesful
150         ", "SETTINGSJS", function (m) {}, function (m) {});
151         successCB ();
152     },
153                                     function (msg)
154     {
155         sap.Logger.debug("Setting Exchange failed" +
156         msg, "SETTINGSJS", function (m) {}, function (m) {});
157         sap.Settings.isInitialized = false;
158         errorCB ();
```



```

155                                     });
156                                     }
157                                     , function () {
158                                     console.log("unlock
failed");
159                                     sap.Logger.debug("unlock failed
", "SETTINGSJS", function(m) {}, function(m) {});
160                                     }
161                                     );
162
163                                     }
164
165
166                                     },
167
168                                     /**
169                                     * Function called by the application to register the
notification types to receive.
170                                     *
171                                     * @param {string} [types] Types of notifications the
application wants to receive. The different types of notifications
are expressed in <code>notificationType</code>
172                                     * @param {string} [successCallback] Success callback to
call when registration is successful.
173                                     * @param {string} [errorCallback] Error callback to call
when registration attempt fails.
174                                     * @param {string} [notificationlistenerfunc] The function
that receives the notification for processing by the application.
175                                     * @param {string} [senderId] The sender ID that is used
for GCM registration. For other platforms it is null.
176                                     * @private
177                                     * @memberof sap.Push
178                                     * @function registerForNotificationTypes
179                                     * @example
180                                     * regid = "211112269206";
181                                     * function registerSuccess(msg) {}

```

```
182     * function registerFailure(mesg) {}
183     * function ProcessNotification(mesg) {}
184     *
sap.Push.registerForNotificationTypes(sap.Push.notificationType.badge | sap.Push.notificationType.sound |
sap.Push.notificationType.alert, registerSuccess, registerFailure,
ProcessNotification, regid);
185     */
186
187     registerForNotification: function (types, successCallback,
errorCallback, notificationListenerFunc, senderId) {
188         if(device.platform == "iPhone" || device.platform ==
"iPad" || device.platform == "iPod touch" || device.platform == "iOS"
|| device.platform == "Android") {
189             sap.Push.RegisterSuccess = successCallback;
190             sap.Push.RegisterFailed = errorCallback;
191             sap.Push.ProcessNotificationForUser =
notificationListenerFunc;
192             sap.Push.call_native(successCallback,
"registerForNotificationTypes", [types, senderId]);
193
194         }
195
196     },
197
198     /* Core APIS */
199
200     /**
201     * Function called by the application to register the
notification types to receive.
202     *
203     * @param {string} types Types of notifications the
application wants to receive. The different types of notifications
are expressed in <code>notificationType</code>
204     *
Notification types allowed are Disable all
notifications (NONE: 0), Set badge count on app icon (BADGE: 1), Play
sounds on receiving notification (SOUNDS: 2) and Show alert on
receiving notification (ALERT: 4).
```

```

205      * @param {string} successCallback Success callback to
call when registration is successful.

206      * @param {string} errorCallback Error callback to call
when registration attempt fails.

207      * @param {string} notificationlistenerfunc The function
that receives the notification for processing by the application.

208      * @param {string} [senderId] The sender ID that is used
for GCM registration. For other platforms it is null.

209      * @public

210      * @memberof sap.Push

211      * @function registerForNotificationTypes

212      * @example

213      * regid = "211112269206";

214      * function registerSuccess(msg){}

215      * function registerFailure(msg) {}

216      * function ProcessNotification(msg){}

217      *
sap.Push.registerForNotificationTypes(sap.Push.notificationType.bad
ge | sap.Push.notificationType.sound |
sap.Push.notificationType.alert, registerSuccess, registerFailure,
ProcessNotification, regid);

218      */

219      registerForNotificationTypes: function (types,
successCallback, errorCallback, notificationListenerFunc, senderId )
{

220          sap.Push.getConnectionSettings(function(){

221              sap.Logger.debug("getting
Connection Settings", "PUSHJS", function(m){}, function(m){});

222              console.log("getting Connection
Settings");

223          sap.Push.registerForNotification(types, successCallback,
errorCallback, notificationListenerFunc, senderId );

224          },

225          function(){});

226      },

227

228

```

```
229      /**
230      * Function called by the application to unregister from
future notifications.
231      *
232      * @param {function} callback Success callback to call
when deregistration is successful. This callback function will
contain a string with a message. This message is for informative
purposes only.
233      * @public
234      * @memberof sap.Push
235      * @function unregisterForNotificationTypes
236      * @example
237      * function unregCallback(msg) {}
238      *
sap.Push.unregisterForNotificationTypes(unregCallback);
239      */
240
241      unregisterForNotificationTypes: function (callbak) {
242          if(device.platform == "iPhone" || device.platform ==
"iPad" || device.platform == "iPod touch" || device.platform == "iOS"
|| device.platform == "Android") {
243              sap.Push.call_native(callbak, "unregisterForNotification");
244          }
245      },
246
247      /**
248      * Used to fetch the badge count for the application. This
function is used by iOS only. Other platforms do not have the badge
count concept.
249      *
250      * @param {function} callback Success callback to call
when to send the badge count. The callback function will contain an
argument in JSON format with the current badge count. Look at the
example for the details on how to use them.
251      * @public
252      * @memberof sap.Push
```

```

253      * @function getBadgeNumber
254      * @example
255      * function getBadgeNumCallback(data) { badgecount =
data["badgecount"]; }
256      * sap.Push.getBadgeNumber(getBadgeNumCallback);
257      */
258      getBadgeNumber: function(callback)
259      {
260          if (sap.Push.isPlatformIOS()) {
261              sap.Push.call_native(callback,
"getBadgeNumber");
262          }
263      },
264
265      /**
266      * Used to set the badge count for the application. This
function is used by iOS only. Other platforms do not have the badge
count concept.
267      *
268      * @param {number} number The badge count to set for the
application.
269      * @param {function} callback Success callback to call
when to send the badge count. The callback function will contain an
argument in string format. This argument can be used for informative
purposes.
270      * @public
271      * @memberof sap.Push
272      * @function setBadgeNumber
273      * @example
274      * function badgeCallback(msg){}
275      * badgenum = 10;
276      * sap.Push.setBadgeNumber(badgenum, badgeCallback);
277      */
278      setBadgeNumber: function (number, callback) {
279          if (sap.Push.isPlatformIOS()) {

```

Kapsel Development

```
280         sap.Push.call_native(callback, "setBadgeNumber",
281         [number]);
282     },
283
284     /**
285     * Used to reset the badge count for the application. This
286     * function is used by iOS only. Other platforms do not have the badge
287     * count concept.
288     *
289     * @param {function} callback Success callback to call
290     * when the badge count is reset. The callback function will contain an
291     * argument in string format. This argument can be used for informative
292     * purpose.
293     * @public
294     * @memberof sap.Push
295     * @function resetBadge
296     * @example
297     * function badgeCallback(msg) {}
298     * sap.Push.resetBadge(badgeCallback);
299     */
300
301     resetBadge: function (callback) {
302         if (sap.Push.isPlatformIOS()) {
303             sap.Push.call_native(callback, "resetBadge");
304         }
305     },
306
307     /**
308     * This method updates the application with the new device
309     * token in the SAP Mobile Platform server.
310     *
311     */
```

```

308     * @param {string} [devtok] The device token received from
the APNS/GCM device registration.
309     * @public
310     * @callback {function} [callback] The callback function
that is called with the registration result.
311     * @memberof sap.Push
312     * @example
313     * function callback(msg) {}
314     * devToken = "123123213213"; //sample device token
315     * sap.Push.updateWithDeviceToken(devToken, callback);
316     */
317
318     updateWithDeviceToken: function (devtok, callback) {
319         if (sap.Push.isPlatformIOS() || device.platform ==
"Android" ) {
320             sap.Push.call_native(callback,
"updateWithDeviceToken", [devtok]);
321         }
322     },
323
324     /**
325     * This method checks for any notifications received while
the application was not running in the foreground. You can call
this
326     * function directly or register with an event handler to
be called automatically. It is okay to call this function even if the
device is not yet registered for push notifications.
327     * @param {function} callback The callback function that
receives the notification. The callback function will receive a
string as its argument. This string will contain the notification
message sent from the server, intact.
328     * @memberof sap.Push
329     * @example
330     * function processBackgroundMessage(msg) {
331     *
332     * }
333     * function checkBackgroundNotification() {

```

```
334      *
sap.Push.checkForNotification(processBackgroudMessage);
335      * }
336      * document.addEventListener("onSapLogonSuccess",
checkBackgroundNotification, false);
337      * document.addEventListener("onSapResumeSuccess",
checkBackgroundNotification, false);
338      **/
339
340      checkForNotification: function(callback) {
341          if (sap.Push.isPlatformIOS() || device.platform ==
"Android" ) {
342              sap.Push.call_native(callback,
"checkForNotification");
343          }
344      },
345
346      /**
347      * This is an internal function, which is called when there
is a push notification.
348      * @private
349      **/
350      ProcessNotification: function(message) {
351          if (sap.Push.ProcessNotificationForUser == null )
352          {
353              console.log("No Processing function provided");
354              sap.Logger.debug("Notification listener function is
not registered. Register it by calling
registerForNotificationTypes", "PUSHJS", function(m) {}, function(m)
{});
355          } else {
356              sap.Push.ProcessNotificationForUser(message);
357          }
358      },
359      /**
```



```

360      * This is an internal function, which is automatically
361      * @private
362      **/
363      initPlugin: function(callback) {
364          if ( device.platform == "Android")
365          {
366              args = [];
367              exec(
368                  callback,
369                  function(){ sap.Logger.debug("Plugin
Initialization","PUSHJS",function(m){},function(m){}); } ,
370                  'SMPPushPlugin',
371                  "initPlugin",
372                  args);
373          }
374      }
375
376      };
377
378
379      /**
380      * Local private variables
381      */
382      module.exports.RegisterSuccess = null;
383      module.exports.RegisterFailed = null;
384      module.exports.ProcessNotificationForUser = null;
385      /**
386      * Enum for types of push notification.
387      * @enum {number}
388      * @private
389      */
390      module.exports.notificationType = {

```

```
391     /** Disable all notifications */
392     NONE: 0,
393     /** Set badge count on app icon */
394     BADGE: 1,
395     /** Play sounds upon receiving notification */
396     SOUNDS: 2,
397     /** Show alert upon receiving notification */
398     ALERT: 4
399 };
400
401
402
403
404     document.addEventListener('deviceready',
405     module.exports.initPlugin, false);
406
```

Using the Kapsel EncryptedStorage Plugin

The EncryptedStorage plugin provides an encrypted local storage mechanism to allow storage of an application's private data on the user's device.

EncryptedStorage Plugin Overview

The Kapsel EncryptedStorage plugin adds an encrypted key/value pair storage option to Cordova, which uses the same API method signature as the browser's local storage option and is non-blocking.

This allows you to store data locally and securely on the device, so that you do not have to retrieve the data from the server every time the application is opened. The user can access and view the data on the device. The data in the encrypted local store is protected by the user's operating system account credentials, so that data cannot be accessed by anyone who is not logged on as the authenticated user, however, the data stored in local storage is not secure against access by other applications run by the authenticated user, so you should not use encrypted local storage to store sensitive information such as digital rights management keys or licensing tokens.

Secure storage is an API based on the w3 Web storage API, interface Storage (<http://www.w3.org/TR/2013/PR-webstorage-20130409/#the-storage-interface>).

Note: On Android, you cannot store more than 1MB for a single key/value pair, as the strings are encoded in UTF-8, which means the maximum length of a complex string that can be successfully stored is less than the maximum length of a string with only simple characters (since simple characters are encoded with a single byte, and complex characters are encoded with up to 4 bytes).

Adding the EncryptedStorage Plugin

Install the the EncryptedStorage plugin using the Cordova command line interface.

Prerequisites

- Set up the development environment.
- Create your Cordova Project.
- Add your OS platforms.

Task

1. Add the EncryptedStorage plugin by entering the following at the command prompt, or terminal:

On Windows:

```
cordova -d plugin add <SDK_HOME>\MobileSDK3\KapselSDK
\plugins\encryptedstorage
```

On Mac:

```
cordova -d plugin add ~<SDK_HOME>/MobileSDK3/KapselSDK/
plugins/encryptedstorage
```

Note: The path you enter to the Kapsel plugin must be the absolute path (not relative path).

2. (Optional) To see a list of installed plugins in your Cordova project, open a command prompt or terminal window, navigate to your Cordova project folder, and enter:

```
cordova plugins
```

The Cordova command line interface returns a JSON array showing installed plugins, for example:

```
[ 'org.apache.cordova.core.camera',
  'org.apache.cordova.core.device-motion',
  'org.apache.cordova.core.file' ]
```

In this example, the Cordova project has the Cordova core Camera, Accelerator (device-motion), and File plugins installed.

3. Modify the files in the `www` folder for the project as necessary, then copy them to the platform directories by running:

```
cordova -d prepare android
cordova -d prepare ios
```

4. Use the Android IDE or Xcode to deploy and run the project.

Note: If you are using an iOS device, remember to add the "clienthubEntitlements" to the Keychain Groups in the Entitlement section in Xcode.

Kapsel EncryptedStorage API Reference

The Kapsel EncryptedStorage API Reference provides usage information for EncryptedStorage API classes and methods, as well as provides sample source code.

EncryptedStorage namespace

The EncryptedStorage class is used as a secure local store.

The EncryptedStorage API is based on the W3C web storage API, but has two major differences: it is asynchronous, and it has a constructor with a password.

Note: There is a security flaw on some versions of Android with the Pseudo Random Number Generation. The first time the native code of this plugin runs it applies the fix for this issue. However, the fix needs to be applied before any use of Java Cryptography Architecture primitives. Therefore, it is a good idea to run this plugin (call a function that has a native component: length, key, getItem, setItem, removeItem, clear) before using any other security-related plugin, to protect yourself against the possibility that the other plugin does not apply this fix. No other Kapsel plugins are affected, so you need not do this on their behalf. For more details about the security flaw, see <http://android-developers.blogspot.com/2013/08/some-securerandom-thoughts.html>

Adding and Removing the EncryptedStorage Plugin

The EncryptedStorage plugin is added and removed using the *Cordova CLI*.

To add the EncryptedStorage plugin to your project, use the following command:

```
cordova plugin add <path to directory containing Kapsel plugins>\encryptedstorage
```

To remove the EncryptedStorage plugin from your project, use the following command:

```
cordova plugin rm com.sap.mp.cordova.plugins.encryptedstorage
```

Members

Name	Description
<i>COMPLEX_STRING_MAXIMUM_LENGTH</i> on page 250	This constant is the length of the largest string that is guaranteed to be successfully stored on Android.
<i>ERROR_BAD_PASSWORD</i> on page 250	This error code indicates that the operation failed due to an incorrect password.
<i>ERROR_GREATER_THAN_MAXIMUM_SIZE</i> on page 251	This error indicates that the string was too large to store.
<i>ERROR_INVALID_PARAMETER</i> on page 251	This error code indicates an invalid parameter was provided. (eg: a string given where a number was required).
<i>ERROR_UNKNOWN</i> on page 251	This error code indicates an unknown error occurred.
<i>SIMPLE_STRING_MAXIMUM_LENGTH</i> on page 251	This constant is the length of the largest string that can successfully be stored on Android.

Methods

Name	Description
<i>clear(successCallback, errorCallback)</i> on page 252	This function removes all items from the store.
<i>getItem(key, successCallback, errorCallback)</i> on page 252	This function gets the value corresponding to the given key.
<i>key(index, successCallback, errorCallback)</i> on page 253	This function gets the key corresponding to the given index.
<i>length(successCallback, errorCallback)</i> on page 254	This function gets the length of the store.
<i>removeItem(key, successCallback, errorCallback)</i> on page 255	This function removes the item corresponding to the given key.
<i>setItem(key, value, successCallback, errorCallback)</i> on page 256	This function sets an item with the given key and value.

Type Definitions

Name	Description
<i>errorCallback(errorCode)</i> on page 257	Callback function that is invoked in case of an error.
<i>getItemSuccessCallback(value)</i> on page 258	
<i>keySuccessCallback(key)</i> on page 258	
<i>lengthSuccessCallback(length)</i> on page 259	
<i>successCallback</i> on page 259	Callback function that is invoked on a successful call to a function that does not need to return anything.

Source

encryptedstorage.js, line 38 on page 261.

COMPLEX_STRING_MAXIMUM_LENGTH member

This constant is the length of the largest string that is guaranteed to be successfully stored on Android.

The limit depends on how many bytes the string takes up when encoded with UTF-8 (under which encoding characters can take up to 4 bytes). This is the maximum length of a string for which every character takes all 4 bytes. Note that this size restriction is present only on Android and not iOS.

Syntax

```
<constant> COMPLEX_STRING_MAXIMUM_LENGTH
```

Source

encryptedstorage.js, line 306 on page 271.

ERROR_BAD_PASSWORD member

This error code indicates that the operation failed due to an incorrect password.

The password is set by the constructor of EncryptedStorage.

Syntax

```
<constant> ERROR_BAD_PASSWORD
```

Source

encryptedstorage.js, line 278 on page 270.

ERROR_GREATER_THAN_MAXIMUM_SIZE member

This error indicates that the string was too large to store.

Only applies to Android. For iOS, no hard limit is imposed, but be aware of device memory constraints.

Syntax

```
<constant> ERROR_GREATER_THAN_MAXIMUM_SIZE
```

Source

encryptedstorage.js, line 286 on page 270.

ERROR_INVALID_PARAMETER member

This error code indicates an invalid parameter was provided. (eg: a string given where a number was required).

Syntax

```
<constant> ERROR_INVALID_PARAMETER
```

Source

encryptedstorage.js, line 270 on page 270.

ERROR_UNKNOWN member

This error code indicates an unknown error occurred.

Syntax

```
<constant> ERROR_UNKNOWN
```

Source

encryptedstorage.js, line 263 on page 270.

SIMPLE_STRING_MAXIMUM_LENGTH member

This constant is the length of the largest string that can successfully be stored on Android.

Only if all the characters in the string are encoded in 1 byte in UTF-8 can a string actually be this big. Since characters in UTF-8 can take up to 4 bytes, if you do not know the contents of a string, the maximum length that is guaranteed to be successful is `EncryptedStorage.COMPLEX_STRING_MAXIMUM_LENGTH`, which is `EncryptedStorage.SIMPLE_STRING_MAXIMUM_LENGTH/4`. Note that this size restriction is present only on Android and not iOS.

Syntax

```
<constant> SIMPLE_STRING_MAXIMUM_LENGTH
```

Source

encryptedstorage.js, line 294 on page 271.

clear(successCallback, errorCallback) method

This function removes all items from the store.

If there are no items in the store in the first place, that is still counted as a success.

Syntax

clear(successCallback, errorCallback)

Parameters

Name	Type	Description
<i>successCallback</i>	<i>sap.EncryptedStorage~successCallback</i> on page 259	If successful, the successCallback is invoked with no parameters.
<i>errorCallback</i>	<i>sap.EncryptedStorage~errorCallback</i> on page 257	If there is an error, the errorCallback is invoked with an <code>ErrorInfo</code> object as the parameter.

Example

```
var store = new sap.EncryptedStorage("storeName", "storePassword");
var successCallback = function() {
    alert("Store cleared!");
}
var errorCallback = function(error) {
    alert("An error occurred: " + JSON.stringify());
}
store.clear(successCallback, errorCallback);
```

Source

encryptedstorage.js, line 229 on page 268.

getItem(key, successCallback, errorCallback) method

This function gets the value corresponding to the given key.

If there is no item with the given key, then the success callback is invoked with null as the parameter.

Syntax

getItem(key, successCallback, errorCallback)

Parameters

Name	Type	Description
<i>key</i>	String	The key of the item for which to get the value. If null or undefined is passed, "null" is used.
<i>successCallback</i>	<i>sap.EncryptedStorage~getItemSuccessCallback</i> on page 258	If successful, the successCallback is invoked with the value as the parameter (or null if the key did not exist).
<i>errorCallback</i>	<i>sap.EncryptedStorage~errorCallback</i> on page 257	If there is an error, the errorCallback is invoked with an ErrorInfo object as the parameter.

Example

```
var store = new sap.EncryptedStorage("storeName", "storePassword");
var successCallback = function(value) {
    alert("Value is " + value);
}
var errorCallback = function(error) {
    alert("An error occurred: " + JSON.stringify(error));
}
store.getItem("theKey", successCallback, errorCallback);
```

Source

encryptedstorage.js, line 118 on page 264.

key(index, successCallback, errorCallback) method

This function gets the key corresponding to the given index.

Syntax

key(index, successCallback, errorCallback)

Parameters

Name	Type	Description
------	------	-------------

<i>index</i>	number	The index of the store for which to get the key. Valid indices are integers from zero (the first index), up to, but not including, the length of the store. If the index is out of bounds, then the success callback is invoked with null as the parameter.
<i>successCallback</i>	<i>sap.EncryptedStorage~keySuccessCallback</i> on page 258	If successful, the successCallback is invoked with the key as the parameter.
<i>errorCallback</i>	<i>sap.EncryptedStorage~errorCallback</i> on page 257	If there is an error, the errorCallback is invoked with an <code>ErrorInfo</code> object as the parameter.

Example

```
// This example shows how to get the key for the last item.
var store = new sap.EncryptedStorage("storeName", "storePassword");
var errorCallback = function( error ){
    alert("An error occurred: " + JSON.stringify(error));
}
var keySuccessCallback = function(key) {
    alert("Last key is " + key);
}
var lengthSuccessCallback = function(length) {
    store.key(length - 1, keySuccessCallback, errorCallback);
}
store.length(lengthSuccessCallback, errorCallback);
```

Source

encryptedstorage.js, line 79 on page 263.

length(successCallback, errorCallback) method

This function gets the length of the store.

The length of a store is the number of key/value pairs that are in the store.

Syntax

length(successCallback, errorCallback)

Parameters

Name	Type	Description
------	------	-------------

<i>successCallback</i>	<i>sap.EncryptedStorage~length-SuccessCallback</i> on page 259	If successful, the successCallback is invoked with the length of the store as the parameter.
<i>errorCallback</i>	<i>sap.EncryptedStorage~error-Callback</i> on page 257	If there is an error, the errorCallback is invoked with an ErrorInfo object as the parameter.

Example

```
var store = new sap.EncryptedStorage("storeName", "storePassword");
var successCallback = function(length) {
    alert("Length is " + length);
}
var errorCallback = function(error) {
    alert("An error occurred: " + JSON.stringify(error));
}
store.length(successCallback, errorCallback);
```

Source

encryptedstorage.js, line 46 on page 261.

removeItem(key, successCallback, errorCallback) method

This function removes the item corresponding to the given key.

If there is no item with the given key in the first place, that is still counted as a success.

Syntax

removeItem(key, successCallback, errorCallback)

Parameters

Name	Type	Description
<i>key</i>	String	The key of the item to remove. If null or undefined is passed, "null" is used.
<i>successCallback</i>	<i>sap.EncryptedStorage~successCallback</i> on page 259	If successful, the successCallback is invoked with no parameters.
<i>errorCallback</i>	<i>sap.EncryptedStorage~error-Callback</i> on page 257	If there is an error, the errorCallback is invoked with an ErrorInfo object as the parameter.

Example

```

var store = new sap.EncryptedStorage("storeName", "storePassword");
var successCallback = function() {
    alert("Value removed");
}
var errorCallback = function(error) {
    alert("An error occurred: " + JSON.stringify(error));
}
store.removeItem("somekey", successCallback, errorCallback);

```

Source

encryptedstorage.js, line 195 on page 267.

setItem(key, value, successCallback, errorCallback) method

This function sets an item with the given key and value.

If no item exists with the given key, then a new item is created. If an item does exist with the the given key, then its value is overwritten with the given value.

Note: On Android there is a size limit on the string to be stored. See *sap.EncryptedStorage#SIMPLE_STRING_MAXIMUM_LENGTH* on page 251 and *sap.EncryptedStorage#COMPLEX_STRING_MAXIMUM_LENGTH* on page 250 for more details.

Syntax

setItem(key, value, successCallback, errorCallback)

Parameters

Name	Type	Description
<i>key</i>	String	The key of the item to set.If null or undefined is passed, "null" is used.
<i>value</i>	String	The value of the item to set.If null or undefined is passed, "null" is used.
<i>successCallback</i>	<i>sap.EncryptedStorage~successCallback</i> on page 259	If successful, the successCallback is invoked with no parameters.

<i>errorCallback</i>	<i>sap.EncryptedStorage~error-Callback</i> on page 257	If there is an error, the error-Callback is invoked with an <code>ErrorInfo</code> object as the parameter.
----------------------	--	---

Example

```
var store = new sap.EncryptedStorage("storeName", "storePassword");
var successCallback = function() {
    alert("Item has been set.");
}
var errorCallback = function(error) {
    alert("An error occurred: " + JSON.stringify(error));
}
store.setItem("somekey", "somevalue", successCallback, errorCallback);
```

Source

encryptedstorage.js, line 154 on page 265.

***errorCallback(errorCode)* type**

Callback function that is invoked in case of an error.

Syntax

errorCallback(errorCode)

Parameters

Name	Type	Description
<i>errorCode</i>	number	An error code indicating what went wrong. Will be one of <i>sap.EncryptedStorage#ERROR_UNKNOWN</i> on page 251, <i>sap.EncryptedStorage#ERROR_INVALID_PARAMETER</i> on page 251, <i>sap.EncryptedStorage#ERROR_BAD_PASSWORD</i> on page 250, or <i>sap.EncryptedStorage#ERROR_GREATER_THAN_MAXIMUM_SIZE</i> on page 251.

Example

```
function errorCallback(errCode) {
    //Set the default error message. Used if an invalid code is passed to the
```

```

//function (just in case) but also to cover the
//sap.EncryptedStorage.ERROR_UNKNOWN case as well.
var msg = "Unkown Error";
switch (errCode) {
  case sap.EncryptedStorage.ERROR_INVALID_PARAMETER:
    msg = "Invalid parameter passed to method";
    break;
  case sap.EncryptedStorage.ERROR_BAD_PASSWORD :
    msg = "Incorrect password";
    break;
  case sap.EncryptedStorage.ERROR_GREATER_THAN_MAXIMUM_SIZE:
    msg = "Item (string) value too large to write to store";
    break;
};
//Write the error to the log
console.error(msg);
//Let the user know what happened
navigator.notification.alert(msg, null, "EncryptedStorage Error",
"OK");
};

```

Source

encryptedstorage.js, line 328 on page 272.

getItemSuccessCallback(value) type

Callback function that is invoked on a successful call to EncryptedStorage.getItem. If the returned value is null, that means the key passed to EncryptedStorage.getItem did not exist.

Syntax

getItemSuccessCallback(*value*)

Parameters

Name	Type	Description
<i>value</i>	String	The value of the item with the given key. Will be null if the key passed to EncryptedStorage.getItem did not exist.

Source

encryptedstorage.js, line 326 on page 272.

keySuccessCallback(key) type

Callback function that is invoked on a successful call to EncryptedStorage.key. If the key returned is null that means the index passed to EncryptedStorage.key was out of bounds.

Syntax

```
keySuccessCallback( key )
```

Parameters

Name	Type	Description
<i>key</i>	String	The key corresponding to the given index. Will be null if the index passed to EncryptedStorage.key was out of bounds.

Source

encryptedstorage.js, line 324 on page 272.

lengthSuccessCallback(length) type

Callback function that is invoked on a successful call to EncryptedStorage.length.

Syntax

```
lengthSuccessCallback( length )
```

Parameters

Name	Type	Description
<i>length</i>	number	The number of key/value pairs in the store.

Source

encryptedstorage.js, line 322 on page 272.

successCallback type

Callback function that is invoked on a successful call to a function that does not need to return anything.

Syntax

```
successCallback()
```

Source

encryptedstorage.js, line 320 on page 272.

Source code

encryptedstorage.js

```
1 // ${project.version}
2 var argscheck = require('cordova/argscheck'),
3     exec = require("cordova/exec");
4
5 /**
6  * The EncryptedStorage class is used as a secure local
7  * store. The EncryptedStorage API is based on the
8  * W3C Web storage API, but has two major differences--it is
9  * asynchronous, and it has a constructor with
10 * a password.<br/>
11 * <br/>
12 * <b>Note:</b> There is a security flaw on some versions of
13 * Android with the Pseudo Random Number Generation.
14 * The first time the native code of this plugin runs it
15 * applies the fix for this issue. However, the
16 * fix needs to be applied before any use of Java Cryptography
17 * Architecture primitives. Therefore, it
18 * is a good idea to run this plugin, for example, call a
19 * function that has a native component such as length, key, getItem,
20 * setItem, removeItem, or clear, before using any other
21 * security-related plugin, to protect yourself
22 * against the possibility that the other plugin does not
23 * apply this fix. This issue affects only the EncryptedStorage
24 * plugin,
25 * so you need not do this for other Kapsel plugins. For more
26 * details about the security flaw, see
27 * <a href="http://android-developers.blogspot.com/2013/08/
28 * some-securerandom-thoughts.html">
29 * http://android-developers.blogspot.com/2013/08/some-
30 * securerandom-thoughts.html</a><br/>
31 * <br/>
32 * <b>Adding and Removing the EncryptedStorage Plugin</b><br/>
33 * <br/>
34 * Add or remove the EncryptedStorage plugin by using the
35 * <a href="http://cordova.apache.org/docs/en/edge/
36 * guide_cli_index.md.html#The%20Command-line%20Interface">Cordova
37 * CLI</a>.<br/>
```



```

23      * <br/>
24      * To add the EncryptedStorage plugin to your project, use the
following command:<br/>
25      * cordova plugin add <path to directory containing Kapsel
plugins>\encryptedstorage<br/>
26      * <br/>
27      * To remove the EncryptedStorage plugin from your project,
use the following command:<br/>
28      * cordova plugin rm
com.sap.mp.cordova.plugins.encryptedstorage
29      * @namespace
30      * @alias EncryptedStorage
31      * @memberof sap
32      * @param {String} storeName The name of the store to create.
All stores with different names
33      * act independently, while stores with the same name (and
same password) act as the same store.
34      * If null or undefined is passed, an empty string is used.
35      * @param {String} password The password of the store to
create. If null or undefined is passed,
36      * an empty string is used.
37      */
38      EncryptedStorage = function (storeName, password) {
39          // private variables
40          var that = this;
41          var storagePassword = password ? password : "";
42          var storageName = storeName ? storeName : "";
43
44          // privileged functions
45
46          /**
47           * This function gets the length of the store. The length
of a store
48           * is the number of key/value pairs that are in the
store.

```

```
49      * @param {sap.EncryptedStorage~lengthSuccessCallback}
successCallback If successful,
50      * the successCallback is invoked with the length of the
store as
51      * the parameter.
52      * @param {sap.EncryptedStorage~errorCallback}
errorCallback If there is an error,
53      * the errorCallback is invoked with an ErrorInfo object as
the parameter.
54      * @memberof sap.EncryptedStorage
55      * @function length
56      * @instance
57      * @example
58      * var store = new sap.EncryptedStorage("storeName",
"storePassword");
59      * var successCallback = function(length) {
60      *     alert("Length is " + length);
61      * }
62      * var errorCallback = function(error) {
63      *     alert("An error occurred: " +
JSON.stringify(error));
64      * }
65      * store.length(successCallback, errorCallback);
66      */
67      this.length = function (successCallback, errorCallback)
{
68          try{
69              argscheck.checkArgs('FF',
'EncryptedStorage.length', arguments);
70          }catch(ex){
71              errorCallback(this.ERROR_INVALID_PARAMETER);
72              return;
73          }
74      }
75      cordova.exec(successCallback, errorCallback,
"EncryptedStorage",
```

```

76         "length", [storageName, storagePassword]);
77     }
78
79     /**
80      * This function gets the key corresponding to the given
81      * index.
82      * @param {number} index The index of the store for which
83      * to get the key.
84      * Valid indices are integers from zero (the first index),
85      * up to, but not including,
86      * the length of the store. If the index is out of bounds,
87      * then the success
88      * callback is invoked with null as the parameter.
89      * @param {sap.EncryptedStorage~keySuccessCallback}
90      * successCallback If successful,
91      * the successCallback is invoked with the key as the
92      * parameter.
93      * @param {sap.EncryptedStorage~errorCallback}
94      * errorCallback If there is an error,
95      * the errorCallback is invoked with an ErrorInfo object as
96      * the parameter.
97      * @memberof sap.EncryptedStorage
98      * @function key
99      * @instance
100     * @example
101     * // This example shows how to get the key for the last
102     * item.
103     * var store = new sap.EncryptedStorage("storeName",
104     * "storePassword");
105     * var errorCallback = function( error ){
106     *     alert("An error occurred: " +
107     * JSON.stringify(error));
108     * }
109     * var keySuccessCallback = function(key) {
110     *     alert("Last key is " + key);
111     * }
112     * var lengthSuccessCallback = function(length) {

```

Kapsel Development

```
102     *     store.key(length - 1, keySuccessCallback,
errorCallback);
103     * }
104     * store.length(lengthSuccessCallback, errorCallback);
105     */
106     this.key = function (index, successCallback,
errorCallback) {
107         try{
108             argscheck.checkArgs('NFF', 'EncryptedStorage.key',
arguments);
109         }catch(ex){
110             errorCallback(this.ERROR_INVALID_PARAMETER);
111             return;
112         }
113
114         cordova.exec(successCallback, errorCallback,
"EncryptedStorage",
115             "key", [storageName, storagePassword, index]);
116     }
117
118     /**
119     * This function gets the value corresponding to the given
key. If there is no
120     * item with the given key, then the success callback is
invoked with null as
121     * the parameter.
122     * @param {String} key The key of the item for which to get
the value. If null or undefined is
123     * passed, "null" is used.
124     * @param {sap.EncryptedStorage~getItemSuccessCallback}
successCallback If successful,
125     * the successCallback is invoked with the value as the
parameter (or null if the key
126     * did not exist).
127     * @param {sap.EncryptedStorage~errorCallback}
errorCallback If there is an error,
```

```
128     * the errorCallback is invoked with an ErrorInfo object
129     * as the parameter.
130     * @memberof sap.EncryptedStorage
131     * @function getItem
132     * @instance
133     * @example
134     * var store = new sap.EncryptedStorage("storeName",
135     * "storePassword");
136     * var successCallback = function(value) {
137     *     alert("Value is " + value);
138     * }
139     * var errorCallback = function(error) {
140     *     alert("An error occurred: " +
141     * JSON.stringify(error));
142     * }
143     * store.getItem("theKey", successCallback,
144     * errorCallback);
145     */
146     this.getItem = function (key, successCallback,
147     errorCallback) {
148         try{
149             argscheck.checkArgs('SFF',
150             'EncryptedStorage.getItem', arguments);
151         }catch(ex){
152             errorCallback(this.ERROR_INVALID_PARAMETER);
153             return;
154         }
155     }
156     cordova.exec(successCallback, errorCallback,
157     "EncryptedStorage",
158     "getItem", [storageName, storagePassword,
159     key]);
160 }
```

Kapsel Development

```
155      * This function sets an item with the given key and
value. If no item exists with
156      * the given key, a new item is created. If an item does
exist with the
157      * the given key, its value is overwritten with the given
value.<br/>
158      * <br/>
159      * Note: On Android there is a size limit on the string to
be stored. See
160      * {@link
sap.EncryptedStorage#SIMPLE_STRING_MAXIMUM_LENGTH} and {@link
sap.EncryptedStorage#COMPLEX_STRING_MAXIMUM_LENGTH}
161      * for more details.
162      * @param {String} key The key of the item to set. If null
or undefined is passed,
163      * "null" is used.
164      * @param {String} value The value of the item to set. If
null or undefined is passed,
165      * "null" is used.
166      * @param {sap.EncryptedStorage~successCallback}
successCallback If successful,
167      * the successCallback is invoked with no parameters.
168      * @param {sap.EncryptedStorage~errorCallback}
errorCallback If there is an error,
169      * the errorCallback is invoked with an ErrorInfo object
as the parameter.
170      * @memberof sap.EncryptedStorage
171      * @function setItem
172      * @instance
173      * @example
174      * var store = new sap.EncryptedStorage("storeName",
"storePassword");
175      * var successCallback = function() {
176      *     alert("Item has been set.");
177      * }
178      * var errorCallback = function(error) {
```

```

179         * alert("An error occurred: " +
JSON.stringify(error));
180     * }
181     * store.setItem("somekey", "somevalue", successCallback,
errorCallback);
182     */
183     this.setItem = function (key, value, successCallback,
errorCallback) {
184         try{
185             argscheck.checkArgs('SSFF',
'EncryptedStorage.setItem', arguments);
186         }catch(ex){
187             errorCallback(this.ERROR_INVALID_PARAMETER);
188             return;
189         }
190
191         cordova.exec(successCallback, errorCallback,
"EncryptedStorage",
192             "setItem", [storageName, storagePassword, key,
value]);
193     }
194
195     /**
196     * This function removes the item corresponding to the
given key. If there is no
197     * item with the given key in the first place, that is
still counted as a success.
198     * @param {String} key The key of the item to remove. If
null or undefined is
199     * passed, "null" is used.
200     * @param {sap.EncryptedStorage~successCallback}
successCallback If successful,
201     * the successCallback is invoked with no parameters.
202     * @param {sap.EncryptedStorage~errorCallback}
errorCallback If there is an error,
203     * the errorCallback is invoked with an ErrorInfo object
as the parameter.

```

Kapsel Development

```
204      * @memberof sap.EncryptedStorage
205      * @function removeItem
206      * @instance
207      * @example
208      * var store = new sap.EncryptedStorage("storeName",
"storePassword");
209      * var successCallback = function() {
210      *     alert("Value removed");
211      * }
212      * var errorCallback = function(error) {
213      *     alert("An error occurred: " +
JSON.stringify(error));
214      * }
215      * store.removeItem("somekey", successCallback,
errorCallback);
216      */
217      this.removeItem = function (key, successCallback,
errorCallback) {
218          try{
219              argscheck.checkArgs('SFF',
'EncryptedStorage.removeItem', arguments);
220          }catch(ex){
221              errorCallback(this.ERROR_INVALID_PARAMETER);
222              return;
223          }
224
225          cordova.exec(successCallback, errorCallback,
"EncryptedStorage",
226              "removeItem", [storageName, storagePassword,
key]);
227      }
228
229      /**
230      * This function removes all items from the store. If there
are no
```



```

231         * items in the store, it is still counted as a
success.
232         * @param {sap.EncryptedStorage~successCallback}
successCallback If successful,
233         * the successCallback is invoked with no parameters.
234         * @param {sap.EncryptedStorage~errorCallback}
errorCallback If there is an error,
235         * the errorCallback is invoked with an ErrorInfo object
as the parameter.
236         * @memberof sap.EncryptedStorage
237         * @function clear
238         * @instance
239         * @example
240         * var store = new sap.EncryptedStorage("storeName",
"storePassword");
241         * var successCallback = function() {
242         *     alert("Store cleared!");
243         * }
244         * var errorCallback = function(error) {
245         *     alert("An error occurred: " + JSON.stringify());
246         * }
247         * store.clear(successCallback, errorCallback);
248         */
249         this.clear = function (successCallback, errorCallback)
{
250             try{
251                 argscheck.checkArgs('FF',
'EncryptedStorage.clear', arguments);
252             }catch(ex){
253                 errorCallback(this.ERROR_INVALID_PARAMETER);
254                 return;
255             }
256
257             cordova.exec(successCallback, errorCallback,
"EncryptedStorage",

```

```
258         "clear", [storageName, storagePassword]);
259     }
260 };
261
262 // Error codes
263 /**
264  * This error code indicates an unknown error occurred.
265  * @memberof sap.EncryptedStorage
266  * @name sap.EncryptedStorage#ERROR_UNKNOWN
267  * @constant
268  */
269 EncryptedStorage.prototype.ERROR_UNKNOWN = 0;
270 /**
271  * This error code indicates an invalid parameter was
272  * provided.
273  * (eg: a string given where a number was required).
274  * @memberof sap.EncryptedStorage
275  * @name sap.EncryptedStorage#ERROR_INVALID_PARAMETER
276  * @constant
277  */
278 EncryptedStorage.prototype.ERROR_INVALID_PARAMETER = 1;
279 /**
280  * This error code indicates that the operation failed due to
281  * an incorrect password. The password is
282  * set by the constructor of {@link EncryptedStorage}.
283  * @memberof sap.EncryptedStorage
284  * @name sap.EncryptedStorage#ERROR_BAD_PASSWORD
285  * @constant
286  */
287 EncryptedStorage.prototype.ERROR_BAD_PASSWORD = 2;
288 /**
289  * This error indicates that the string was too large to
290  * store. This error applies only to Android.
```

```

288     * For iOS, no hard limit is imposed, but be aware of device
memory constraints.
289     * @memberof sap.EncryptedStorage
290     * @name
sap.EncryptedStorage#ERROR_GREATER_THAN_MAXIMUM_SIZE
291     * @constant
292     */
293     EncryptedStorage.prototype.ERROR_GREATER_THAN_MAXIMUM_SIZE =
3;
294     /**
295     * This constant is the length of the largest string that can
successfully be stored on Android. Only if all the
296     * characters in the string are encoded in 1 byte in UTF-8 can
a string actually be this big. Since
297     * characters in UTF-8 can take up to 4 bytes, if you do not
know the contents of a string, the maximum
298     * length that is guaranteed to be successful is {@link
EncryptedStorage.COMPLEX_STRING_MAXIMUM_LENGTH}, which is
299     * {@link EncryptedStorage.SIMPLE_STRING_MAXIMUM_LENGTH}/4.
This size restriction is present only on
300     * Android.
301     * @memberof sap.EncryptedStorage
302     * @name sap.EncryptedStorage#SIMPLE_STRING_MAXIMUM_LENGTH
303     * @constant
304     */
305     EncryptedStorage.prototype.SIMPLE_STRING_MAXIMUM_LENGTH =
1048542;
306     /**
307     * This constant is the length of the largest string that is
guaranteed to be successfully stored on Android. The
308     * limit depends on how many bytes the string takes up when
encoded with UTF-8 (under which encoding
309     * characters can take up to 4 bytes). This is the maximum
length of a string for which every character
310     * takes all 4 bytes. This size restriction is present only
on Android.
311     * @memberof sap.EncryptedStorage

```

Kapsel Development

```
312     * @name sap.EncryptedStorage#COMPLEX_STRING_MAXIMUM_LENGTH
313     * @constant
314     */
315     EncryptedStorage.prototype.COMPLEX_STRING_MAXIMUM_LENGTH =
316     262135;
317     module.exports = EncryptedStorage;
318
319
320     /**
321     * Callback function that is invoked on a successful call to a
322     * function that does
323     *
324     * not need to return anything.
325     *
326     * @callback sap.EncryptedStorage~successCallback
327     */
328     * Callback function that is invoked on a successful call to
329     * {@link EncryptedStorage.length}.
330     *
331     * @callback sap.EncryptedStorage~lengthSuccessCallback
332     *
333     * @param {number} length The number of key/value pairs in the
334     * store.
335     */
336     * Callback function that is invoked on a successful call to
337     * {@link EncryptedStorage.key}.
338     *
339     * If the key returned is null, that means the index passed to
340     * {@link EncryptedStorage.key} was out of bounds.
341     *
342     * @callback sap.EncryptedStorage~keySuccessCallback
```

```

340      *
341      * @param {String} key The key corresponding to the given
342      * index. Will be null if the index passed to
343      * {@link EncryptedStorage.key} was out of bounds.
344      */
345     /**
346     * Callback function that is invoked on a successful call to
347     * {@link EncryptedStorage.getItem}.
348     * If the returned value is null, that means the key passed to
349     * {@link EncryptedStorage.getItem} did not exist.
350     *
351     * @callback sap.EncryptedStorage~getItemSuccessCallback
352     *
353     * @param {String} value The value of the item with the given
354     * key. Will be null if the key passed to
355     * {@link EncryptedStorage.getItem} did not exist.
356     */
357     /**
358     * Callback function that is invoked in case of an error.
359     *
360     * @callback sap.EncryptedStorage~errorCallback
361     *
362     * @param {number} errorCode An error code indicating what
363     * went wrong. Will be one of {@link
364     * sap.EncryptedStorage#ERROR_UNKNOWN},
365     * {@link sap.EncryptedStorage#ERROR_INVALID_PARAMETER},
366     * {@link sap.EncryptedStorage#ERROR_BAD_PASSWORD}, or
367     * {@link
368     * sap.EncryptedStorage#ERROR_GREATER_THAN_MAXIMUM_SIZE}.
369     *
370     * @example
371     * function errorCallback(errCode) {
372     *     //Set the default error message. Used if an invalid code
373     *     is passed to the

```

```
367      *      //function (just in case) but also to cover the
368      *      //sap.EncryptedStorage.ERROR_UNKNOWN case as well.
369      *      var msg = "Unkown Error";
370      *      switch (errCode) {
371      *          case sap.EncryptedStorage.ERROR_INVALID_PARAMETER:
372      *              msg = "Invalid parameter passed to method";
373      *              break;
374      *          case sap.EncryptedStorage.ERROR_BAD_PASSWORD :
375      *              msg = "Incorrec password";
376      *              break;
377      *          case
378      * sap.EncryptedStorage.ERROR_GREATER_THAN_MAXIMUM_SIZE:
379      *              msg = "Item (string) value too large to write to
380      * store";
381      *              break;
382      *      };
383      *      //Write the error to the log
384      *      console.error(msg);
385      *      //Let the user know what happened
386      *      navigator.notification.alert(msg, null,
387      * "EncryptedStorage Error", "OK");
388      * };
```

Using the Kapsel Settings Plugin

Use the Settings plugin to trigger an operation with the SAP Mobile Platform Server that allows an application to store device and user settings for later use.

Settings Plugin Overview

The Settings plugin exchanges application connection settings with the server settings.

If application settings such as log level and log upload mode are changed on the server, the Settings plugin synchronizes the information with the Kapsel application. Since some of that information is used in the Push plugin, the Push plugin requires the Settings plugin.

The APIs for the Settings plugin allow device and user settings to be stored on the device to make a connection with the SAP Mobile Platform Server. The client sends the server the

DeviceType, DeviceModel, PushEnabled, and other push-related statuses. The settings also use the device token that is received during device registration. The server uses this information to determine whether to send a GCM or APNS push notification.

Adding the Settings Plugin

To install the Settings plugin, use the Cordova command line interface.

Prerequisites

- Set up the development environment.
- Create your Cordova Project.
- Add your OS platforms.

Task

The Settings plugin has dependencies on the Logger plugin, so when you install the Settings plugin, the Logger plugin is added automatically.

1. Add the Settings plugin by entering the following at the command prompt, or terminal:

On Windows:

```
cordova -d plugin add <SDK_HOME>\MobileSDK3\KapselSDK
\plugins\settings
```

On Mac:

```
cordova -d plugin add ~<SDK_HOME>/MobileSDK3/KapselSDK/
plugins/settings
```

Note: The path you enter to the Kapsel plugin must be the absolute path (not relative path).

2. (Optional) To see a list of installed plugins in your Cordova project, open a command prompt or terminal window, navigate to your Cordova project folder, and enter:

```
cordova plugins
```

The Cordova command line interface returns a JSON array showing installed plugins, for example:

```
[ 'org.apache.cordova.core.camera',
'org.apache.cordova.core.device-motion',
'org.apache.cordova.core.file' ]
```

In this example, the Cordova project has the Cordova core Camera, Accelerator (device-motion), and File plugins installed.

3. Modify the files in the www folder for the project as necessary, then copy them to the platform directories by running:

```
cordova -d prepare android
cordova -d prepare ios
```

- Use the Android IDE or Xcode to deploy and run the project.

Note: If you are using an iOS device, remember to add the "clienthubEntitlements" to the Keychain Groups in the Entitlement section in Xcode.

Kapsel Settings API Reference

The Kapsel Settings API Reference provides usage information for Settings API classes and methods, as well as provides sample source code.

Settings namespace

Provides settings exchange functionality

Methods

Name	Description
<i>start(connectionData, successCallback, errorCallback)</i> on page 276	Starts the settings exchange.

Source

settings.js, line 12 on page 277.

start(connectionData, successCallback, errorCallback) method

Starts the settings exchange.

Syntax

<static> *start(connectionData, successCallback, errorCallback)*

Parameters

Name	Type	Description
<i>connectionData</i>	String	This example below shows the structure of the connection data.
<i>successCallback</i>	function	Function to invoke if the exchange is successful.
<i>errorCallback</i>	function	Function to invoke if the exchange failed.

Example

```
connectionData = {
  "keyMAFLogonOperationContextConnectionData": {
    "keyMAFLogonConnectionDataApplicationSettings":
    {
      "DeviceType":device.platform,

```



```

    "DeviceModel":device.model,
    "ApplicationConnectionId":"yourappconnectionid"
  },
  "keyMAFLogonConnectionDataBaseURL":"servername:port"
},
"keyMAFLogonOperationContextApplicationId":"yourapplicationid",
"keyMAFLogonOperationContextBackendUserName":"yourusername",
"keyMAFLogonOperationContextBackendPassword":"password",
"keyMAFLogonOperationContextSecurityConfig":"securityConfigName",
"keySSEnabled":keySSEnabled
};
sap.Settings.start(connectionData, function(msg) {

    sap.Logger.debug("Setting Exchange
is successful "+msg,"SMP_SETTINGS_JS",function(m){},function(m){});
    },
    function(msg) {
    sap.Logger.debug("Setting Exchange
failed" + msg,"SMP_SETTINGS_JS",function(m){},function(m){});
    });

```

Source

settings.js, line 96 on page 281.

Source code***settings.js***

```

1      // ${project.version}
2      var exec = require("cordova/exec");
3
4
5      /**
6       * The Kapsel Settings plugin provides settings exchange
7       * functionality.
8       *
9       * @namespace
10      * @alias Settings
11      * @memberof sap
12      */
13
14      var SettingsExchange = function () {};
15
16      SettingsExchange.prototype.connectionData = null;
17      SettingsExchange.prototype.store = null;

```

```
16     SettingsExchange.prototype.settingsSuccess = null;
17     SettingsExchange.prototype.SettingsError = null;
18     SettingsExchange.prototype.isInitialized = false;
19
20
21     /**
22     * Starts the settings exchange process upon onSapLogonSuccess
23     event.
24     * @private
25     */
26     var doSettingExchange = function () {
27
28         sap.Settings.isInitialized = true;
29         var pd = "";
30         sap.Logon.unlock(function (connectionInfo) {
31             var userName =
32 connectionInfo["registrationContext"]["user"];
33             var password =
34 connectionInfo["registrationContext"]["password"];
35             var applicationConnectionId =
36 connectionInfo["applicationConnectionId"];
37             var securityConfig =
38 connectionInfo["registrationContext"]["securityConfig"];
39             var endpoint =
40 connectionInfo["applicationEndpointURL"];
41             var keySSLEnabled = "false";
42             var splitendpoint =
43 endpoint.split("/");
44             if (splitendpoint[0] == "https:")
45             {
46                 keySSLEnabled="true";
47             }
48             if (securityConfig == null) {
49                 securityConfig = "";
```

```
44         }
45         var burl = splitendpoint[2];
46         var appId = splitendpoint[3];
47         pd = appId+userName+password;
48         sap.Settings.store = new
sap.EncryptedStorage("SettingsStore", pd);
49         connectionData = {
50
51         "keyMAFLogonOperationContextConnectionData": {
52
53         "keyMAFLogonConnectionDataApplicationSettings":
54
55         {
56
57         "DeviceType":device.platform,
58
59         "DeviceModel":device.model,
60
61         "ApplicationConnectionId":applicationConnectionId
62
63         },
64
65         "keyMAFLogonConnectionDataBaseURL":burl
66
67         },
68
69         "keyMAFLogonOperationContextApplicationId":appId,
70
71         "keyMAFLogonOperationContextBackendUserName":userName,
72
73         "keyMAFLogonOperationContextBackendPassword":password,
74
75         "keyMAFLogonOperationContextSecurityConfig":securityConfig,
76
77         "keySSEnabled":keySSEnabled
78
79         };
80
81         sap.Settings.start(connectionData,
82
83         function(msg) {
84
85         sap.Settings.isInitialized = true;
```

```
68 sap.Logger.debug("Setting Exchange is successful
"+mesg, "SMP_SETTINGS_JS", function(m) {}, function(m) {});
69                                     },
70                                     function(mesg) {
71 sap.Logger.debug("Setting Exchange failed" +
mesg, "SMP_SETTINGS_JS", function(m) {}, function(m) {});
72 sap.Settings.isInitialized = false;
73                                     });
74                                     }
75                                     , function () {
76                                     sap.Logger.debug("unlock failed
", "SMP_SETTINGS_JS", function(m) {}, function(m) {});
77                                     }
78                                     );
79
80
81     };
82
83     document.addEventListener("onSapLogonSuccess",
doSettingExchange, false);
84     document.addEventListener("onSapResumeSuccess",
doSettingExchange, false);
85
86     SettingsExchange.prototype.reset = function(key, sucessCB,
errorCB)
87     {
88         if ((typeof(sap.Settings.store) != undefined) &&
(sap.Settings.store != null)) {
89             sap.Settings.store.removeItem(key, sucessCB,
errorCB);
90         } else {
91             errorCB("Cannot access setting store");
92         }
93     }
```

```
94
95
96     /**
97     * Starts the settings exchange.
98     * @public
99     * @memberof sap.Settings
100    * @method start
101    * @param {String} connectionData The example below shows the
    structure of the connection data.
102    * @param {function} successCallback Function to invoke if the
    exchange is successful.
103    * @param {function} errorCallback Function to invoke if the
    exchange failed.
104    * @example
105    * connectionData = {
106    *     "keyMAFLogonOperationContextConnectionData": {
107    *     "keyMAFLogonConnectionDataApplicationSettings":
108    *     {
109    *     "DeviceType":device.platform,
110    *     "DeviceModel":device.model,
111    *     "ApplicationConnectionId":"yourappconnectionid"
112    *     },
113    *     "keyMAFLogonConnectionDataBaseURL":"servername:port"
114    *     },
115    *     "keyMAFLogonOperationContextApplicationId":"yourapplicationid",
116    *     "keyMAFLogonOperationContextBackendUserName":"yourusername",
117    *     "keyMAFLogonOperationContextBackendPassword":"password",
118    *     "keyMAFLogonOperationContextSecurityConfig":"securityConfigName",
119    *     "keySSEEnabled":keySSEEnabled
120    *     };
```

```
121     * sap.Settings.start(connectionData, function(msg) {
122     *
123     *
124     * sap.Logger.debug("Setting Exchange is successful
125     * "+msg, "SMP_SETTINGS_JS", function(m) {}, function(m) {});
126     *                                     },
127     *                                     function(msg) {
128     *                                     sap.Logger.debug("Setting
129     * Exchange failed" + msg, "SMP_SETTINGS_JS", function(m) {}, function(m)
130     * {});
131     *                                     });
132     */
133     SettingsExchange.prototype.start = function (connectionData,
134     successCallback, errorCallback) {
135         sap.Settings.settingsSuccess = successCallback;
136         sap.Settings.SettingsError = errorCallback;
137         sap.Settings.connectionData = connectionData;
138         sap.Logger.debug("Accessing the data from
139         vault", "SMP_SETTINGS_JS", function(m) {}, function(m) {});
140         sap.Settings.store.getItem("settingsdata",
141         sap.Settings.getStoreDataSuccess, sap.Settings.getStoreDataError);
142     };
143
144     /**
145     * This is a private function. End user will not use this
146     * plugin directly.
147     * This function gets called after the start function is able
148     * to read the current settings from the secured storage.
149     * @private
```

```
148     * @param {String} value This is the value of the current
setting exchange stored in the secured store.
149     **/
150
151     SettingsExchange.prototype.getStoreDataSuccess =
function(value) {
152         storedSettings = value;
153         sap.Logger.debug("Exchanging the
data", "SMP_SETTINGS_JS", function(m) {}, function(m) {});
154         exec(sap.Settings.SettingsExchangeDone,
155             sap.Settings.SettingsExchangeError,
156             "SMPSettingsExchangePlugin",
157             "start",
[JSON.stringify(connectionData), storedSettings]);
158     }
159
160     /**
161     * This is a private function. End user will not use this
plugin directly.
162     * This function is called after the start function is unable
to read the current settings from the secured storage.
163     * @private
164     * @param {String} message This is the error message produced
by the encrypted storage.
165     **/
166     SettingsExchange.prototype.getStoreDataError =
function(message) {
167         sap.Logger.debug("Setting exchange failed to read data
store: Proceeding without data", function(m) {}, function(m) {});
168     }
169
170
171     /**
172     * This is a private function. End user will not use this
plugin directly.
173     * This function is called after the settings exchange
completes successfully.
```

Kapsel Development

```
174      * @private
175      * @param {String} message This is the message produced when
the settings plugin completes successfully.
176      **/
177
178      SettingsExchange.prototype.SettingsExchangeDone =
function(message) {
179          sap.Logger.debug("Setting Exchange
Success", "SMP_SETTINGS_JS", function(m) {}, function(m) {});
180          var jsondata = JSON.parse(message);
181          settingsString = JSON.stringify(jsondata["data"]);
182          sap.Settings.store.setItem("settingsdata", settingsString, sap.Settin
gs.SettingsWriteDone, sap.Settings.SettingsWriteError);
183          if (sap.Settings.settingsSuccess != null) {
184              sap.Logger.debug("Setting exchange
successful", "SMP_SETTINGS_JS", function(m) {}, function(m) {});
185              sap.Settings.settingsSuccess(jsondata["msg"]);
186          }
187      }
188
189      /**
190      * This is a private function. End user will not use this
plugin directly.
191      * This function is called after the settings exchange
completes succefully
192      * @private
193      * @param {String} message This is the error message produced
when the settings plugin has an error.
194      **/
195      SettingsExchange.prototype.SettingsExchangeError =
function(message) {
196          sap.Logger.error("Setting Exchange failed calling the
error callback funciton", "SMP_SETTINGS_JS", function(m) {}, function(m)
{});
197          if (sap.Settings.SettingsError != null) {
198              sap.Settings.SettingsError(message);
```



```
199     }
200 }
201
202 /**
203  * This is a private function. End user will not use this
plugin directly.
204  * This function is called after the setting data is stored
successfully.
205  * @private
206  * @param {String} message This is the message produced upon
successful storing of settings to the encrypted store.
207  */
208 SettingsExchange.prototype.SettingsWriteDone =
function(message) {
209     sap.Logger.debug("Setting
stored", "SMP_SETTINGS_JS", function(m) {}, function(m) {});
210
211 }
212
213 /**
214  * This is a private function. End user will not use this
plugin directly.
215  * This function is called after the storing of the setting
data fails.
216  * @private
217  * @param {String} message This is the message produced upon
failure to store the settings to the encrypted store.
218  */
219 SettingsExchange.prototype.SettingsWriteError =
function(message) {
220     sap.Logger.error("Setting store
failed", "SMP_SETTINGS_JS", function(m) {}, function(m) {});
221 }
222
223 /**
224  * This is a private function. End user will not use this
plugin directly.
```

```
225     * This function is called after the deviceready. This
226     * uploads the logs to the server.
227     * @private
228     * @param {boolean} uploadLog This indicates whether the
229     * upload log is currently enabled or disabled.
230     **/
231     SettingsExchange.prototype.logLevelUpdated =
232     function(logLevel)
233     {
234         sap.Logger.setLogLevel(logLevel,
235         sap.Settings.LogLevelSetSuccess, sap.Settings.LogLevelSetFailed);
236         sap.Logger.upload(sap.Settings.logUploadedSuccess,
237         sap.Settings.logUploadFailed);
238     }
239
240 /**
241     * This is a private function. End user will not use this
242     * plugin directly.
243     * This function is called when the log upload succeeds.
244     * @private
245     * @param {mesg} logupload message
246     **/
247     SettingsExchange.prototype.LogLevelSetSuccess =
248     function(mesg) {
249         sap.Logger.debug("Log level set
250         successful", "SMP_SETTINGS_JS", function(m) {}, function(m) {});
251     }
252
253 /**
254     * This is a private function. End user will not use this
255     * plugin directly.
256     * This function is called when the log upload succeeds.
257     * @private
258     * @param {mesg} logupload message
259     **/
260     SettingsExchange.prototype.LogLevelSetFailed =
261     function(mesg) {
```

```
251     sap.Logger.error("Log level set
failed", "SMP_SETTINGS_JS", function(m){}, function(m){});
252   }
253
254   /**
255    * This is a private function. End user will not use this
plugin directly.
256    * This function is called when the log upload succeeds.
257    * @private
258    * @param {mesg} logupload message
259    */
260   SettingsExchange.prototype.logUploadedSuccess =
function(mesg) {
261     sap.Logger.debug("Log upload
successful", "SMP_SETTINGS_JS", function(m){}, function(m){});
262   }
263   /**
264    * This is a private function. End user will not use this
plugin directly.
265    * This function is called when the log upload fails.
266    * @private
267    * @param {mesg} logupload failure message
268    */
269   SettingsExchange.prototype.logUploadFailed = function(mesg)
{
270     sap.Logger.error("upload log
failed", "SMP_SETTINGS_JS", function(m){}, function(m){});
271
272   }
273
274
275
276   module.exports = new SettingsExchange();
277
278
```

279

280

281

Developing a Kapsel Application With OData Online

Creating an OData Application

Create an OData application in the Management Cockpit.

1. In the Management Cockpit home page, click Settings to define your application's security settings.
2. In the Edit Security Profile dialog, click **New**.
3. Enter a name for your security profile and optional description.
4. In the Authentication Providers section, click **Add**.
5. Choose an authentication provider and click **Create**.
6. In the Management Cockpit Home page, click Applications.
7. Click **New**.
8. In the New Application window, enter the values for your application:

Field	Value
ID	Unique identifier for the application in reverse domain notation.
Name	Application name.
Vendor	(Optional) Vendor who developed the application.
Version	Application version. Currently, only version 1.0 is supported.
Type	Application type. <ul style="list-style-type: none"> • Native – native iOS and Android applications. • Hybrid – container-based applications, such as Kapsel. • Agency – meta data-driven applications, such as Agency. Application configuration options differ depending on your selection.
Description	(Optional) Short description of the application.

9. Click **Save**.

Application-related tabs appear, and you are ready to configure the application, based on the application type.

10. In the Backend page, enter the end point for the back end system to which the application will connect. For example, `http://localhost:8090/odata`.

11. In the bottom right-hand corner, click **Save**.
12. Run the application on the device or emulator, and click **Register**.
13. In the registration page, enter the values, and click **Register**.

The user name and password combination should have permission to access the OData backend.

- 14.

Creating an Application Connection

You must explicitly register the application connection using SAP Mobile Platform.

You can specify customized application properties for the client with the request. Provide the application connection ID, X-SMP-APPID, using an explicit request header or a cookie. If the value is missing, SAP Mobile Platform generates a universally unique ID (UUID), which is communicated to the device through the response cookie X-SMP-APPID.

Create an anonymous or authenticated application connection by issuing a POST request to this URL, including the application connection properties:

```
http[s]://<host:port>/[public]/odata/applications/{latest|v1}/
{appid}/Connections
```

The URL contains these components:

- **host** – the host is defined by host name and should match with the domain registered with SAP Mobile Platform. If the requested domain name does not match, default domain is used..
- **port** – the port for listening to OData-based requests. By default the port number is 8080.
- **public** – if included, an anonymous connection is allowed.
- **odata/applications/** – refers to the OData services associated with the application resources.
- **{latest|v1}** – version of the service document.
- **appid** – name of the application.
- **Connections** – name of the OData collection.

Application connection properties are optional. You can create an application connection without including any application properties.

DeviceType is an application connection property that you may set. Valid values for DeviceType are:

- **Android** – Android devices.
- **iPhone** – Apple iPhone.
- **iPad** – Apple iPad.
- **iPod** – Apple iPod.
- **iOS** – iOS devices.

- **Blackberry** – Blackberry devices.
- **Windows** – includes desktop or servers with Windows OS, such as Windows XP, Windows Vista, Windows 7, and Windows Server series OS.
- **WinPhone8** – includes Windows mobile.
- **Windows8** – includes Windows desktop version.

Specifying any other value for DeviceType returns a value "Unknown" in the DeviceType column.

Example of creating an application connection

Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:m="http://
schemas.microsoft.com/ado/2007/08/dataservices/metadata"
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
  <category term="applications.Connection" scheme="http://
schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  <content type="application/xml">
    <m:properties>
      <d:AndroidGcmRegistrationId>398123745023</d:
AndroidGcmRegistrationId>
    </m:properties>
  </content>
</entry>
```

Response

```
<?xml version="1.0" encoding="utf-8"?>
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:m="http://
schemas.microsoft.com/ado/2007/08/dataservices/metadata"
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
xml:base="https://<smp base URL>/odata/applications/latest/
e2eTest/">
  <id>https://<application URL>/odata/applications/latest/e2eTest/
Connections ('4891dd0f-0735-47cc-a599-76bf8a16d457')</id>
  <title type="text" />
  <updated>2012-10-19T09:05:25Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Connection"
href="Connections ('4891dd0f-0735-47cc-a599-76bf8a16d457') " />
  <category term="applications.Connection" scheme="http://
schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <content type="application/xml">
    <m:properties>
      <d:ETag>2012-10-19 14:35:24.0</d:ETag>
      <d:ApplicationConnectionId>4891dd0f-0735-47cc-
a599-76bf8a16d457</d:ApplicationConnectionId>
      <d:AndroidGcmPushEnabled m:type="Edm.Boolean">false</
d:AndroidGcmPushEnabled>
      <d:AndroidGcmRegistrationId>398123745023</d:
AndroidGcmRegistrationId>
```

```

<d:AndroidGcmSenderId />
<d:ApnsPushEnable m:type="Edm.Boolean">>false</d:ApnsPushEnable>
<d:ApnsDeviceToken />
<d:ApplicationVersion>1.0</d:ApplicationVersion>
<d:BlackberryPushEnabled m:type="Edm.Boolean">>false</
d:BlackberryPushEnabled>
<d:BlackberryDevicePin m:null="true" />
<d:BlackberryBESListenerPort m:type="Edm.Int32">0</
d:BlackberryBESListenerPort>
<d:BlackberryPushAppID m:null="true" />
<d:BlackberryPushBaseURL m:null="true" />
<d:BlackberryPushListenerPort m:type="Edm.Int32">0</
d:BlackberryPushListenerPort>
<d:BlackberryListenerType m:type="Edm.Int32">0</
d:BlackberryListenerType>
<d:CustomizationBundleId />
<d:CustomCustom1 />
<d:CustomCustom2 />
<d:CustomCustom3 />
<d:CustomCustom4 />
<d:DeviceModel m:null="true" />
<d:DeviceType>Unknown</d:DeviceType>
<d:DeviceSubType m:null="true" />
<d:DevicePhoneNumber m:null="true" />
<d:DeviceIMSI m:null="true" />
<d>PasswordPolicyEnabled m:type="Edm.Boolean">>false</
d>PasswordPolicyEnabled>
<d>PasswordPolicyDefaultPasswordAllowed
m:type="Edm.Boolean">>false</d>PasswordPolicyDefaultPasswordAllowed>
<d>PasswordPolicyMinLength m:type="Edm.Int32">0</
d>PasswordPolicyMinLength>
<d>PasswordPolicyDigitRequired m:type="Edm.Boolean">>false</
d>PasswordPolicyDigitRequired>
<d>PasswordPolicyUpperRequired m:type="Edm.Boolean">>false</
d>PasswordPolicyUpperRequired>
<d>PasswordPolicyLowerRequired m:type="Edm.Boolean">>false</
d>PasswordPolicyLowerRequired>
<d>PasswordPolicySpecialRequired m:type="Edm.Boolean">>false</
d>PasswordPolicySpecialRequired>
<d>PasswordPolicyExpiresInNDays m:type="Edm.Int32">0</
d>PasswordPolicyExpiresInNDays>
<d>PasswordPolicyMinUniqueChars m:type="Edm.Int32">0</
d>PasswordPolicyMinUniqueChars>
<d>PasswordPolicyLockTimeout m:type="Edm.Int32">0</
d>PasswordPolicyLockTimeout>
<d>PasswordPolicyRetryLimit m:type="Edm.Int32">0</
d>PasswordPolicyRetryLimit>
<d:ProxyApplicationEndpoint>http://<backend URL></
d:ProxyApplicationEndpoint>
<d:ProxyPushEndpoint>http[s]://<host:port>/Push</
d:ProxyPushEndpoint>
<d:MpnsChannelURI m:null="true" />
<d:WnsChannelURI m:null="true" />
</m:properties>
</content>
</entry>

```

CORS Support

Cross-domain HTTP requests are requests for resources from a different domain than the domain of the resource making the request. Cross-Origin Resource Sharing (CORS) mechanism provides a way for web servers to support cross-site access controls, which enable secure cross-site data transfers.

Getting Application Settings

You can retrieve application connection settings for the device application instance by issuing the GET method.

You can retrieve application settings by either explicitly specifying the application connection ID, or by having the application connection ID determined from the call context (that is, from either the X-SMP-APPCID cookie or X-SMP-APPCID HTTP header, if specified). On the first call, you can simplify your client application code by having the application connection ID determined from the call context, since you have not yet received an application connection ID.

If you supply an application connection ID, perform an HTTP GET request at:

```
http[s]://<host:port>/[public/]odata/applications/{latest|v1}/
{appid}/Connections('{appid}')
```

Response

```
<?xml version='1.0' encoding='utf-8'?>
<entry xmlns="http://www.w3.org/2005/Atom"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/
dataservices/metadata"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/
dataservices"
  xml:base="https://<smp base URL>/odata/applications/v1/
e2eTest/">
  <id>http://https://mobilesmpdev.netweaver.ondemand.com/smp/odata/
applications/v1/e2eTest/
Connections('c9d8a9da-9f36-4ae5-9da5-37d6d90483b5')</id>
  <title type="text" />
  <updated>2012-06-28T09:55:48Z</updated>
  <author><name /></author>
  <link rel="edit" title="Connections"
href="Connections('c9d8a9da-9f36-4ae5-9da5-37d6d90483b5') " />
  <category term="applications.Connection" scheme="http://
schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <content type="application/xml">
    <m:properties>
      <d:ETag m:type="Edm.DateTime">2012-06-28T17:55:47.685</d:ETag>

<d:ApplicationConnectionId>c9d8a9da-9f36-4ae5-9da5-37d6d90483b5</
d:ApplicationConnectionId>
      <d:AndroidGcmPushEnabled m:type="Edm.Boolean">>false</
d:AndroidGcmPushEnabled>
      <d:AndroidGcmRegistrationId m:null="true" />
```



```

    <d:AndroidGcmSenderId m:null="true" />
    <d:ApnsPushEnable m:type="Edm.Boolean">true</d:ApnsPushEnable>
    <d:ApnsDeviceToken m:null="true" />
    <d:ApplicationVersion m:null="true" />
    <d:BlackberryPushEnabled m:type="Edm.Boolean">true</
d:BlackberryPushEnabled>
    <d:BlackberryDevicePin>00000000</d:BlackberryDevicePin>
    <d:BlackberryBESListenerPort m:type="Edm.Int32">5011</
d:BlackberryBESListenerPort>
    <d:BlackberryPushAppID m:null="true" />
    <d:BlackberryPushBaseUrl m:null="true" />
    <d:BlackberryPushListenerPort m:type="Edm.Int32">0</
d:BlackberryPushListenerPort>
    <d:BlackberryListenerType m:type="Edm.Int32">0</
d:BlackberryListenerType>
    <d:CustomCustom1>custom1</d:CustomCustom1>
    <d:CustomCustom2 m:null="true" />
    <d:CustomCustom3 m:null="true" />
    <d:CustomCustom4 m:null="true" />
    <d:DeviceModel m:null="true" />
    <d:DeviceType>Unknown</d:DeviceType>
    <d:DeviceSubType m:null="true" />
    <d:DevicePhoneNumber>12345678901</d:DevicePhoneNumber>
    <d:DeviceImsi m:null="true" />
    <d>PasswordPolicyEnabled m:type="Edm.Boolean">true</
d>PasswordPolicyEnabled>
    <d>PasswordPolicyDefaultPasswordAllowed
m:type="Edm.Boolean">>false</d>PasswordPolicyDefaultPasswordAllowed>
    <d>PasswordPolicyMinLength m:type="Edm.Int32">8</
d>PasswordPolicyMinLength>
    <d>PasswordPolicyDigitRequired m:type="Edm.Boolean">>false</
d>PasswordPolicyDigitRequired>
    <d>PasswordPolicyUpperRequired m:type="Edm.Boolean">>false</
d>PasswordPolicyUpperRequired>
    <d>PasswordPolicyLowerRequired m:type="Edm.Boolean">>false</
d>PasswordPolicyLowerRequired>
    <d>PasswordPolicySpecialRequired m:type="Edm.Boolean">>false</
d>PasswordPolicySpecialRequired>
    <d>PasswordPolicyExpiresInNDays m:type="Edm.Int32">0</
d>PasswordPolicyExpiresInNDays>
    <d>PasswordPolicyMinUniqueChars m:type="Edm.Int32">0</
d>PasswordPolicyMinUniqueChars>
    <d>PasswordPolicyLockTimeout m:type="Edm.Int32">0</
d>PasswordPolicyLockTimeout>
    <d>PasswordPolicyRetryLimit m:type="Edm.Int32">20</
d>PasswordPolicyRetryLimit>
    <d:ProxyApplicationEndpointm:null="true" />
    <d:ProxyPushEndpoint>http://xxue-desktop:8080/GWC/
SMPNotification</d:ProxyPushEndpoint>
    <d:WnsChannelURI m:null="true" />
    <d:MpnsChannelURI m:null="true" />
    <d:WnsPushEnable m:type="Edm.Boolean">>false</d:WnsPushEnable>
    <d:MpnsPushEnable m:type="Edm.Boolean">true</d:MpnsPushEnable>
  </m:properties>
</content>
</entry>

```

You can also retrieve a property value by appending the property name in the URL. For example, to retrieve the `ClientLogLevel` property value, enter:

```
http[s]://<host:port>/[public/]odata/applications/{v1|latest}/  
{appid}/Connections('{appid}')/ClientLogLevel
```

Running and Testing Kapsel Applications

Test your Cordova project by opening it in its respective development environment (Eclipse with Android plugins or Xcode), then run it in the corresponding emulator (Android) or simulator (iOS),

You can launch the emulator or simulator from the Cordova command line interface, or from the development environment.

Client-side Debugging

Debug the Kapsel application on the device or by using a desktop browser.

Debugging in a Desktop Browser

Debug the JavaScript code running in a desktop browser.

This procedure shows how to debug using Chrome. See <https://developers.google.com/chrome-developer-tools/>. In some cases, debugging on the device is necessary, for example, when you debug touch, or code that includes JavaScript files from Apache Cordova or Kapsel, since these expect to run on a mobile device or simulator.

1. In the Chrome menu, choose **Tools > Developer Tools**.
2. Click **Sources** to open a source file.
3. Set break points to step through the code.
4. Use the **Network** tab to examine the OData URL sent and the values received.

Debugging on iOS

This procedure demonstrates how to debug an app that includes Apache Cordova and Kapsel plugins.

This procedure requires a device or simulator running iOS 6 and a Mac that has Safari 6.

1. Connect the device to the Mac with a USB cable, or start the simulator.
2. On the device or simulator, go to **Settings > Safari > Advanced > Web Inspector**, and turn it to **On**.
3. On the device or simulator, open your Kapsel app or a Web page in the Safari browser.
4. On the Mac, in Safari, choose **Develop > iPhone Simulator > index.html**.

Running the Kapsel Application on Android

Open your Cordova based Kapsel project in Eclipse and run it on the emulator.

1. In a Command Prompt window, make sure you are in the project folder and execute the command:

```
cordova prepare android
```
2. Start Eclipse.
3. From the menu, choose **File > Import**.
4. In the Import window, select **Android > Existing Android Code Into Workspace**.
5. Browse to your project, `<ProjectName>/platforms/android`, select the android folder, and click **Open**.
6. Click **Finish**.
The project is imported into Eclipse.
7. Right-click the project node and select **Run As > Android Application**.

Running the Kapsel Application on iOS

Open your project in Xcode and run the application on the simulator.

1. In a terminal window, make sure you are in the project folder and execute the command:

```
cordova prepare ios
```
2. Open Xcode.
3. In a Finder window, browse to your Cordova project folder, `<Project Name>/platforms/ios`.
4. Double-click the `<ProjectName>.xcodproj` file to open the project in Xcode.
5. Select your Simulator type and click the **Run** button.

Package and Deploy Kapsel Applications

Use the Android IDE or Xcode to package the Kapsel app, then use the Management Cockpit to upload the app to the server.

Generating and Uploading Kapsel App Files Using the Command Line Interface

The Kapsel command line interface provides a way to generate a ZIP file that contains the HTML files that make up the app.

Kapsel Development

1. Open a command prompt window, or terminal, and navigate to the folder that contains the Kapsel command line interface, for example:

On Windows:

```
SDK_HOME\MobileSDK3\KapselSDK\cli
```

On Mac:

```
~SDK_HOME/MobileSDK3/KapselSDK/cli
```

2. Run the command:

```
npm -g install
```

On Mac, you may need to run the command as sudo:

```
sudo npm -g install
```

3. Change directories to the directory containing the project and run the command:

```
kapsel package
kapsel deploy <com.mycompany.app_ID> localhost:<port>
<Admin_user_name> <Admin_password>
```

You can, optionally, enter a platform in the package command, such as android or ios. The parameters to the deploy command are the app ID, the SAP Mobile Platform Server host name, and the user ID and password for Management Cockpit.

The ZIP file containing the HTML files that make up the app is generated and then uploaded to SAP Mobile Platform Server, and the Management Cockpit shows that revision *x* was uploaded.

Changing the Default Port

By default, the Kapsel command line interface is configured to use port 8083, as this is the default used for Management Cockpit when installing SAP Mobile Platform Server.

If port 8083 is in use during installation, the installer automatically assigns a different port and notifies you. If the port number changes from 8083, you can change it using the command line.

1. Open a command prompt window, or terminal.
2. Specify the server parameter in the format `server:port`, for example:

```
kapsel deploy <com.mycompany.app_ID> localhost:<port>
<Admin_user_name> <Admin_password>
```

This example shows how to deploy the Kapsel application called "com.sample.app" using port 8084.

```
kapsel deploy com.sample.app localhost:8084 smpAdmin
s3pAdmin
```

Preparing the Application for Upload to the Server

Upload the Kapsel app to SAP Mobile Platform Server.

1. In the Android IDE or Xcode, right-click the project's **www** folder, and compress the items to package the files in a ZIP file.
2. Log in to the Management Cockpit to upload the app.

Uploading and Deploying Hybrid Apps

Upload a new or updated version of a hybrid app package, and deploy it to make the current version available to users. The older version is retained until you delete it.

Prerequisites

The application developer creates the hybrid app package that:

- Contains the contents of the application's **www** folder and `config.xml` of the project, with a separate folder in the archive for each mobile platform (`android/www` and/or `ios/www` in all lower case). Format structure for hybrid apps:

```
| - android
| | - config.xml
| | - www
| - iOS
...

```

- Is compressed into a standard `.zip` file for upload.

Task

1. From Management Cockpit, select **Applications > App Specific Settings**.
2. Under Upload, click **Browse**.
 - a) In the dialog, navigate to the directory.
 - b) Select the hybrid app package, and confirm.

New version information appears for the uploaded Kapsel app for each mobile platform. You cannot change this information.

Property	Description
Required Kapsel Version	Identifies the Kapsel SDK version used to develop the Kapsel app, for example 3.0.0. Note: This version attribute is informational only, and is not used by SAP Mobile Platform Server to determine whether device clients should receive the Web application update.
Development Version	Identifies the internal development version used to develop the Kapsel app.
Description	Describes the Kapsel app.

Property	Description
Revision	Identifies the production version revision. For a newly uploaded Kapsel app, this is blank. <hr/> Note: When the Kapsel app is deployed, the revision number is incremented.

3. When ready, deploy the application:
 - a. Click **Deploy** and confirm to deploy an application to one mobile platform.
 - b. Click **Deploy All** and confirm to deploy the application to all available mobile platforms.

Deployed Kapsel app information appears as the current version and the revision number is incremented.

For device application users:

- When a device user with a default version (revision = 0) of the Kapsel app connects to the server, the server downloads the full Kapsel app.
- When a device user with a version (revision = 1 or higher) of the Kapsel app connects to the server, the server calculates the difference between the user's version and the new version, and downloads a patch containing only the required changes.
- If the application implements the AppUpdate plugin, the server checks for updates when the application starts-up or is resumed.
- If the developer enables a feature in the application, an application or a user can check for updates manually.

4. When ready, delete the older version of an application:
 - a. Click **Remove** and confirm to remove an older version of an application from one mobile platform.
 - b. Click **Remove All** and confirm to remove all older versions of an application from all available mobile platforms.

Deploying Hybrid Apps Using the REST API

Deploy a new or updated hybrid app to SAP Mobile Platform Server using the deploy application REST API.

Once the application is deployed, it is considered to be a new version. You can make it the current version using the promotion REST API. After the application is promoted, users can download a patch to upgrade the application on their devices.

Note: It is not possible to deploy a hybrid app for a specific platform: everything in the file is deployed. Once the application is deployed, you can promote or delete hybrid apps for specific platforms as needed.

Syntax

Perform a POST request to the following URI:

```
https://<host>:<admin_port>/Admin/kapsel/jaxrs/KapselApp/{APP_ID}
```

Parameters

- **file** – The file that contains the application archive, sent as multipart/form-data.

Returns

A response providing information about the new and current version of the application. For example:

```
{ "newVersion":
  { "requiredKapselVersion": "1.5",
    "developmentVersion": "1.2.5",
    "description": "An update for the sample app.",
    "revision": -1},
  "currentVersion":
  { "requiredKapselVersion": "1.5",
    "developmentVersion": "1.2.4",
    "description": "A sample app.",
    "revision": 2}
}
```

On successful deployment, the client receives a 201 status code; otherwise, an HTTP failure code and message.

Examples

Note: This example uses the `curl` command line client and the `--cacert` flag. Your client may require you to pass other arguments or set specific configuration options.

- **Deploy application to all platforms**

```
curl --user <user>:<password> --cacert <your-server.pem> --form
"file=@C:\work\appl.zip" https://localhost:8083/Admin/kapsel/
jaxrs/KapselApp/MyTestAppId
```

Removing Kapsel Plugins

Remove Kapsel plugins from the application.

To remove a plugin, refer to it by the same identifier that appears in the Cordova plugin listing. These steps show an example of how you would remove the logger plugin.

Note: Due to a known Apache issue, **plugin remove** does not currently work properly with Kapsel plugins. See <https://issues.apache.org/jira/browse/CB-41>. Shared dependencies with other plugins may also be removed, leaving the application in a bad state. Instead of removing

plugins, SAP recommends that you start from a clean state and add only the plugins you require.

1. Open a command prompt window and navigate to the Cordova project's directory.
2. (Optional) To see a list of installed plugins, enter:

```
cordova plugins
```

3. Enter:

```
cordova plugin remove <plugin_name>
```

For example, to remove the Logger plugin, enter:

```
cordova plugin remove com.sap.mp.cordova.plugins.logger
```


Index

A

- addEventListener
 - method 25
- Android SDK 3
- Apache Cordova 1
- Apache Cordova Command Line Interface
 - installing 3
- APNs 212
- App ID 213
- applicationId
 - member 49
- applications 297
 - configuring applications 7
 - creating an application definition 7
- AppUpdate 17
 - namespace 23
- AppUpdate plugin
 - adding 22
 - overview 17
- appupdate.js
 - source file 31
- authentication
 - application, defining 11
- AuthProxy
 - namespace 126
- AuthProxy plugin
 - adding 122
 - client certificate 125
 - keytool utility 125
 - overview 122
 - register with SAP Mobile Platform Server 125
 - sap.AuthProxy.CertificateFromLogonManager("clientKey") 125
- authproxy.js
 - source file 145

C

- callback
 - type 230
- CertificateFromFile
 - class 129
- CertificateFromLogonManager
 - class 131

- CertificateFromStore
 - class 132
- changePassword
 - method 51
- checkForNotification
 - method 225
- checking
 - event 27
- class
 - CertificateFromFile 129
 - CertificateFromLogonManager 131
 - CertificateFromStore 132
- clear
 - method 252
- Client Hub 46
- client logs
 - examples 182
 - viewing 182
- clienthubEntitlements keychain group 46
- COMPLEX_STRING_MAXIMUM_LENGTH
 - member 250
- config.xml 13
- configuring applications 7
- Cordova config.xml file 15
- core
 - member 50
- creating an application definition 7
- creating application endpoint URL 9
- creating back-end connection 9

D

- debug
 - method 188
- debugging 294
 - in a desktop browser 294
- default port
 - changing 296
- defining
 - application authentication 11
- deleteCertificateFromStore
 - method 137
- deleteCertificateSuccessCallback
 - type 143
- deploying
 - hybrid app REST API 298

Index

downloading
 event 28

E

EncryptedStorage 246
 namespace 248
EncryptedStorage plugin
 adding 247
encryptedstorage.js
 source file 260
ERR_CERTIFICATE_ALIAS_NOT_FOUND
 member 132
ERR_CERTIFICATE_FILE_NOT_EXIST
 member 133
ERR_CERTIFICATE_INVALID_FILE_FORMAT
 member 133
ERR_CLIENT_CERTIFICATE_VALIDATION
 member 133
ERR_FILE_CERTIFICATE_SOURCE_UNSUPPORTED
 member 133
ERR_GET_CERTIFICATE_FAILED
 member 134
ERR_HTTP_TIMEOUT
 member 134
ERR_INVALID_PARAMETER_VALUE
 member 134
ERR_LOGON_MANAGER_CERTIFICATE_METHOD_NOT_AVAILABLE
 member 134
ERR_LOGON_MANAGER_CORE_NOT_AVAILABLE
 member 135
ERR_MISSING_PARAMETER
 member 135
ERR_NO_SUCH_ACTION
 member 135
ERR_SERVER_CERTIFICATE_VALIDATION
 member 136
ERR_SERVER_REQUEST_FAILED
 member 136
ERR_SYSTEM_CERTIFICATE_SOURCE_UNSUPPORTED
 member 136
ERR_UNKNOWN
 member 136
error
 event 28
 method 189

ERROR_BAD_PASSWORD
 member 250
ERROR_GREATER_THAN_MAXIMUM_SIZE
 member 251
ERROR_INVALID_PARAMETER
 member 251
ERROR_UNKNOWN
 member 251
errorCallback
 type 59, 143, 257
event
 checking 27
 downloading 28
 error 28
 noupdate 29
 updateready 29

F

files
 source 30, 65, 145, 197, 230, 259, 277

G

GCM 214
generateODataHttpClient
 method 138
get
 method 51, 139
getBadgeNumber
 method 226
getItem
 method 252
getItemSuccessCallback
 type 258
getLogLevel
 method 191
getSuccessCallback
 type 60
Git
 installing 3
Google Cloud Messaging Service 214

H

hybrid apps
 deploying, REST API reference 298

I

- info
 - method 192
- init
 - method 52

J

- Java SDK
 - adding to PATH 3
 - installing 3

K

- Kapsel apps
 - similar to hybrid apps 297
 - uploading new version 297
- Kapsel command line interface
 - generating files 295
- key
 - method 253
- keySuccessCallback
 - type 258

L

- length
 - method 254
- lengthSuccessCallback
 - type 259
- life cycle
 - Kapsel apps 297
- lock
 - method 56
- Logger
 - namespace 184
- Logger plugin 178
- logger.js
 - source file 197
- Logger#DEBUG
 - member 187
- Logger#ERROR
 - member 187
- Logger#INFO
 - member 188
- Logger#WARN
 - member 188
- Logon
 - namespace 47

- Logon plugin
 - configuring default values 45
 - installing 43
- Logon.core.deleteRegistration() 46
- LogonController.js
 - source file 66

M

- managePasscode
 - method 56
- member
 - applicationId 49
 - COMPLEX_STRING_MAXIMUM_LENGTH 250
 - core 50
 - ERR_CERTIFICATE_ALIAS_NOT_FOUND 132
 - ERR_CERTIFICATE_FILE_NOT_EXIST 133
 - ERR_CERTIFICATE_INVALID_FILE_FORMAT 133
 - ERR_CLIENT_CERTIFICATE_VALIDATION 133
 - ERR_FILE_CERTIFICATE_SOURCE_UNSUPPORTED 133
 - ERR_GET_CERTIFICATE_FAILED 134
 - ERR_HTTP_TIMEOUT 134
 - ERR_INVALID_PARAMETER_VALUE 134
 - ERR_LOGON_MANAGER_CERTIFICATE_METHOD_NOT_AVAILABLE 134
 - ERR_LOGON_MANAGER_CORE_NOT_AVAILABLE 135
 - ERR_MISSING_PARAMETER 135
 - ERR_NO_SUCH_ACTION 135
 - ERR_SERVER_CERTIFICATE_VALIDATION 136
 - ERR_SERVER_REQUEST_FAILED 136
 - ERR_SYSTEM_CERTIFICATE_SOURCE_UNSUPPORTED 136
 - ERR_UNKNOWN 136
 - ERROR_BAD_PASSWORD 250
 - ERROR_GREATER_THAN_MAXIMUM_SIZE 251
 - ERROR_INVALID_PARAMETER 251
 - ERROR_UNKNOWN 251
 - Logger#DEBUG 187
 - Logger#ERROR 187
 - Logger#INFO 188

Index

- Logger#WARN 188
- SIMPLE_STRING_MAXIMUM_LENGTH 251
- method
 - addEventListener 25
 - changePassword 51
 - checkForNotification 225
 - clear 252
 - debug 188
 - deleteCertificateFromStore 137
 - error 189
 - generateODataHttpClient 138
 - get 51, 139
 - getBadgeNumber 226
 - getItem 252
 - getLogLevel 191
 - info 192
 - init 52
 - key 253
 - length 254
 - lock 56
 - managePasscode 56
 - registerForNotificationTypes 227
 - reloadApp 26
 - removeEventListener 26
 - removeItem 255
 - reset 27
 - resetBadge 228
 - sendRequest 141
 - set 57
 - setBadgeNumber 229
 - setItem 256
 - setLogLevel 193
 - showRegistrationData 58
 - start 276
 - unlock 59
 - unregisterForNotificationTypes 229
 - update 27
 - upload 195
 - warn 196
- N**
- namespace
 - AppUpdate 23
 - AuthProxy 126
 - EncryptedStorage 248
 - Logger 184
 - Logon 47
 - Push 224
 - Settings 276
- Node.js
 - installing 3
- noupdate
 - event 29
- P**
- Plugman 215
- provisioning file 214
- proxy server
 - configuring for Git 3
 - configuring for npm 3
- Push
 - namespace 224
- push.js
 - source file 231
- R**
- registerForNotificationTypes
 - method 227
- reloadApp
 - method 26
- removeEventListener
 - method 26
- removeItem
 - method 255
- reset
 - method 27
- resetBadge
 - method 228
- REST API reference
 - hybrid app, deploying 298
- S**
- sap.Logon.init() 46
- sendRequest
 - method 141
- set
 - method 57
- setBadgeNumber
 - method 229
- setItem
 - method 256
- setLogLevel
 - method 193

- Settings
 - namespace 276
- Settings plugin 274
 - adding 275
 - overview 274
- settings.js
 - source file 277
- showRegistrationData
 - method 58
- SIMPLE_STRING_MAXIMUM_LENGTH
 - member 251
- source
 - files 30, 65, 145, 197, 230, 259, 277
- source file
 - appupdate.js 31
 - authproxy.js 145
 - encryptedstorage.js 260
 - logger.js 197
 - LogonController.js 66
 - push.js 231
 - settings.js 277
- start
 - method 276
- successCallback
 - type 60, 144, 259
- successCallbackNoParameters
 - type 65

T

- type
 - callback 230

- deleteCertificateSuccessCallback 143
- errorCallback 59, 143, 257
- getItemSuccessCallback 258
- getSuccessCallback 60
- keySuccessCallback 258
- lengthSuccessCallback 259
- successCallback 60, 144, 259
- successCallbackNoParameters 65

U

- unlock
 - method 59
- unregisterForNotificationTypes
 - method 229
- update
 - method 27
- updateready
 - event 29
- upload
 - method 195
- uploading
 - Kapsel apps 297

W

- warn
 - method 196

