AT&T Toolkit for Salesforce Platform

Developer Guide

AT&T Developer Program

12/24/2012

Legal Disclaimer

This document and the information contained herein (collectively, the "**Information**") is provided to you (both the individual receiving this document and any legal entity on behalf of which such individual is acting) ("**you**" and "**your**") by AT&T, on behalf of itself and its affiliates ("**AT&T**") for informational purposes only. AT&T is providing the Information to you because AT&T believes the Information may be useful to you. The Information is provided to you solely on the basis that you will be responsible for making your own assessments of the Information and are advised to verify all representations, statements and information before using or relying upon any of the Information. Although AT&T has exercised reasonable care in providing the Information to You, AT&T does not warrant the accuracy of the Information and is not responsible for any damages arising from your use of or reliance upon the Information. You further understand and agree that AT&T in no way represents, and You in no way rely on a belief, that AT&T is providing the Information in accordance with any standard or service (routine, customary or otherwise) related to the consulting, services, hardware or software industries.

AT&T DOES NOT WARRANT THAT THE INFORMATION IS ERROR-FREE. AT&T IS PROVIDING THE INFORMATION TO YOU "AS IS" AND "WITH ALL FAULTS." AT&T DOES NOT WARRANT, BY VIRTUE OF THIS DOCUMENT, OR BY ANY COURSE OF PERFORMANCE, COURSE OF DEALING, USAGE OF TRADE OR ANY COLLATERAL DOCUMENT HEREUNDER OR OTHERWISE, AND HEREBY EXPRESSLY DISCLAIMS, ANY REPRESENTATION OR WARRANTY OF ANY KIND WITH RESPECT TO THE INFORMATION, INCLUDING, WITHOUT LIMITATION, ANY REPRESENTATION OR WARRANTY OF DESIGN, PERFORMANCE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, OR ANY REPRESENTATION OR WARRANTY THAT THE INFORMATION IS APPLICABLE TO OR INTEROPERABLE WITH ANY SYSTEM, DATA, HARDWARE OR SOFTWARE OF ANY KIND. AT&T DISCLAIMS AND IN NO EVENT SHALL BE LIABLE FOR ANY LOSSES OR DAMAGES OF ANY KIND, WHETHER DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, SPECIAL OR EXEMPLARY, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF GOODWILL, COVER, TORTIOUS CONDUCT OR OTHER PECUNIARY LOSS, ARISING OUT OF OR IN ANY WAY RELATED TO THE PROVISION, NON-PROVISION, USE OR NON-USE OF THE INFORMATION, EVEN IF AT&T HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES OR DAMAGES.

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

Table of Contents

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

1. OAuth

The OAuth service provides a safe and secure way for subscribers to access AT&T network resources through a third-party developed application without the risk of compromising security.

Each service has its authorization scope. Only one authorization type can be used for each scope (see paragraph 1.3 for details). If you want to use one authorization object for multiple services, specify the list of authorization scopes.

NOTE: All scopes in the list must have the same authorization type.

The OAuth service consists of the following APIs:

- *AttAuthorization* is the base interface for all authorization types providing common methods to obtain authorization data.
- *AttOAuth* implements common logic to obtain and refresh access token.
- *AttClientCredentialsAuthorization* implements the AT&T Client Credentials Authorization Model used for the AT&T services, such as SMS and MMS, where there is no specific subscriber authorization or consent required.
- *AttCodeAuthorization* implements the AT&T Authorization Code Model used for the AT&T services, such as Location and Device Capabilities, where the subscriber authorization is required.
- *AttOAuthResult* specifies the storage of the AT&T OAuth service results.
- *AttConsentWorkflow* implements the AT&T OAuth workflow to obtain the end-user authorization.
- *AttConsentExtWorkflow* implements the extended workflow to obtain the AT&T end-user authorization used for controllers where user input parameters, such as attachments and large data, cannot be passed through the end-user authorization.

1.1. Creating Client Credentials Authorization

Use client credential authorization when subscriber authorization is not required.

To use client credentials authorization:

1. Create an instance of the *AttClientCredentialsAuthorization* class with the scope of all the required services.

```
ServiceConstants.ScopeType[] SCOPE = new ServiceConstants.ScopeType[]
{ServiceConstants.ScopeType.WAPPUSH, ServiceConstants.ScopeType.MMS,
ServiceConstants.ScopeType.SMS};
AttClientCredentialsAuthorization auth = new
AttClientCredentialsAuthorization(SCOPE);
```

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

NOTE: Client credential authorization can only be used with services that do not require specific subscriber authorization or consent.

2. Create a new service with the authorization object.

```
AttWAPPush wapPush = new AttWAPPush(auth);
...
```

NOTE: Only services specified in the authorization scope can be used.

1.2. Creating Code Authorization

1.2.1. Getting Authorization from the User

You can get user authorization using the AT&T End-User Authorization workflow. It's one of the steps which allows creating OAuth Code Authorization (see paragraph 1.2.2 for more details).

You can use either *AttConsentWorkflow* or *AttConsentExtWorkflow*, or implement consent authorization workflow by yourself (see AT&T OAuth 2.0 service Get End User Authorization documentation).

Use *AttConsentWorkflow* unless user input data, such as attachments and large data sets, cannot be passed through redirection to the AT&T OAuth server. Otherwise, use *AttConsentExtWorkflow*.

To use AttConsentWorkflow, perform the following steps:

1. Create your own controller that extends the *AttConsentWorkflow* class.

```
public class MyController extends AttConsentWorkflow {
        ...
}
```

2. Implement abstract methods of the *AttConsentWorkflow* class.

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

```
protected override void initialize() {
   // initialize controller fields and services here
}

protected override ServiceConstants.ScopeType[] getConsentScope() {
   // return array of controller services scopes where subscriber
   // authorization is required.
}

protected override Map<String, String> getParams() {
   // return map of parameters and user input data which should be
   // passed through redirect to AT&T OAuth server
}

protected override void loadParams(Map<String, String> params) {
   // load parameters passed through redirect to AT&T OAuth server
   // here
}

public override void execute() {
   // execute controller services and action here
}
```

To use *AttConsentExtWorkflow*, perform the following steps:

3. Create your own controller that extends the *AttConsentExtWorkflow* class.

```
public class MyController extends AttConsentExtWorkflow {
        ...
}
```

4. Implement abstract methods of the *AttConsentExtWorkflow* class.

```
protected override void initialize() {
   // initialize controller fields and services here
}

protected override ServiceConstants.ScopeType[] getConsentScope() {
   // return scopes array of controller services which require
   // subscriber authorization.
}

public override void execute() {
   // invoke services and actions of your controller here
}
```

NOTE: The best way to use authorization is to enable controllers on the apex page when the application is authorized.

```
...
<apex:pageBlockButtons>
```

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

```
      <apex:commandButton action="{!authorize}" value="Authorize"
disabled="{!authorized}"/>
      <apex:commandButton action="{!invokeService}" value="Send Message"
disabled="{! not authorized}"/>
</apex:pageBlockButtons>
...
```

1.2.2. Creating Code Authorization

Use code authorization when subscriber authorization is required.

To use code authorization:

1. Create an instance of the *AttCodeAuthorization* class.

```
AttCodeAuthorization auth = new AttCodeAuthorization();
```

2. Get authorization from the user (see paragraph 1.2.1 for more details).
3. Set the authorization code to the value returned by the user authorization. The scope of all the required services is defined in the request to the AT&T OAuth server.

```
auth.setAuthorizationCode('AtDQkEf3G8gojYFJMTLr8lwFQ');
```

4. Create an instance of the required services with the authorization object.

NOTE: Code authorization can only be used for services where the subscriber authorization is required.

```
AttLocation locationService = new AttLocation(auth);
```

1.3. Authorization scopes

Service name	Scope	Authorization type
SMS (see chapter 2 for details)	*ServiceConstants.ScopeType. SEND_SMS*	Client Credentials Authorization (see paragraph 1.1 for details)
MMS (see chapter 3 for details)	*ServiceConstants.ScopeType. SEND_MMS*	Client Credentials Authorization (see paragraph 1.1 for details)
Speech Service (see	*ServiceConstants.ScopeType.*	Client Credentials

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

chapter 4 for details)	*SPEECH*	Authorization (see paragraph 1.1 for details)
WAP Push Service (see chapter 5 for details)	*ServiceConstants.ScopeType. WAPPUSH*	Client Credentials Authorization (see paragraph 1.1 for details)
Device Capabilities (see chapter 6 for details)	ServiceConstants.ScopeType. DEVICE_CAPABILITIES	Code Authorization (see paragraph 1.2.2 for details)
Location Service (see chapter 7 for details)	*ServiceConstants.ScopeType. TL*	Code Authorization (see paragraph 1.2.2 for details)
Payment (see chapter 8 for details)	*ServiceConstants.ScopeType. PAYMENT*	Client Credentials Authorization (see paragraph 1.1 for details)

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

2. SMS Service

The SMS service enables authorized applications to send SMS messages to devices within the AT&T network, receive SMS messages by polling or enabling a listener, and retrieve the delivery status of sent SMS messages.

The SMS service contains the following classes:

- *AttSMSOutbox* for sending SMS messages
- *AttSMSInbox* for receiving SMS messages
- *AttSMSInboxStatus* for getting SMS inbox information
- *AttSMS* for creating SMS objects
- *AttSMSOutboxStatus* for getting SMS delivery information
- *AttSMSCallbackProcessor* for processing received messages immediately
- *AttMessageDeliveryStatus* for getting SMS message delivery status

2.1. Sending SMS Messages

To send an SMS, perform the following steps:

1. Create an instance of the *AttSMSOutbox* class with an *AttAuthorization* object.

```
AttSMSOutbox smsOutbox = new AttSMSOutbox(auth);
```

2. Create an instance of the *AttSMS* class.

```
AttSMS sms = new AttSMS();
```

3. Set the properties of the SMS object.

```
sms.phoneNumber = '4258028620';

sms.messageText = 'hello, world';
```

4. Pass the *AttSMS* object to the *SendMessage* method of the *AttSMSOutbox* object.

```
smsOutbox.sendMessage(sms);
```

2.2. Handling SMS Delivery Data

The *sendMessage* method returns an *AttSMSOutboxStatus* object that contains the following properties:

- *id*, which specifies the unique identifier for the message

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

- *resourceUrl*, which specifies the URI that contains the network delivery status of the sent message

To process the SMS delivery information, call the *getDeliveryStatus* method to receive the status of the SMS message.

```
AttMessageDeliveryStatus answerSMSStatus =
smsStatus.getDeliveryStatus();
```

The *AttMessageDeliveryStatus* class contains the *deliveryInfoList* property, which is a list of *AttMessageDeliveryStatus*.*DeliveryInfo* objects with the following properties:

- *id*, which specifies the unique identifier for the message destination
- *address*, which specifies the mobile number of the receiver
- *deliveryStatus*, which specifies the status of the SMS message that is delivered

2.3. Receiving SMS Messages

To receive SMS messages, perform the following steps:

1. Create an instance of the *AttSMSInbox* class with the short code and the *AttAuthorization* object.

```
AttSMSInbox smsInbox = new AttSMSInbox('123456', auth);
```

NOTE: The class has to be initialized with the short code, which is obtained when you register your app with AT&T.

2. Call the *checkMessages* method.

```
AttSMSInboxStatus inboxMessagesAnswer = smsInbox.checkMessages();
```

The returned *AttSMSInboxStatus* object contains the *inboundSMSMessageList* property with the following properties:

- *inboundSmsMessageList*, which is a list of AttSMSInboxStatus.*InboundSmsMessage* objects
- *NumberOfMessagesInThisBatch*, which specifies the number of messages sent in the (batch) response
- *TotalNumberOfPendingMessages*, which specifies the total number of pending messages that are yet to be retrieved from the server for the given short code
- ResourceURL, which specifies the link to https URL of the API with the message ID

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

2.4. Receiving SMS Messages Using Callbacks

To process received messages immediately using a callback, perform the following steps:

1. Create a class that inherits from the *AttSMSCallbackProcessor* class. Implement the *processMessage* method which is called when a message is received. The *AttSMSCallbackProcessor* class contains the protected *smsRecord* property which is an *AttSMSCallbackProcessor*.AttSMSRecord object that contains the following properties with the details about the received message:

   - *SenderAddress,* which specifies the phone number of the sender with “tel” prefix, such as tel:16309700001
   - *DestinationAddress,* which specifies the short code where the message is sent
   - *MessageId,* which specifies the unique identifier generated by the AT&T system for the received message
   - *Message,* which specifies the text message that the end user sent
   - *MsgDateTime,* which specifies the time and date when the message is received, such as *04-15-2011T12:00:00*

2. Implement the apex REST service which handles HTTP POST method. This method should create an instance of the class that inherits from the *AttSMSCallbackProcessor* class, passing the string representation of the request body as a constructor parameter and call the *processMessage* method.

As a result, you will see the following code:

```
@RestResource(urlMapping='/SMSInboxCallback')

          global class DemoSMSInboxCallback {

                    @HttpPost

                    global static void doPost() {

                              DemoSMSToEmailCallbackProcessor callbackProcessor = new

                                        RestContext.request.requestBody.toString());

                              callbackProcessor.processMessage();

                    }

          }
```

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

3. Create a separate site on *SalesForce* platform and activate it if it is not available. Enable apex class access to this site.

NOTE: Access should be enabled to the class implementing the REST service. The service will be available at *https://base address of the site***/services/apexrest/***url mapping of service.*

4. In the *SMS Mobile Originated URI* settings of the AT&T account, enter the web-address of the REST service. After that, the REST service will be called each time the message is sent to this short code number.

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

3. MMS

The MMS service enables authorized applications to send MMS messages to mobile devices within the AT&T network, receive messages and retrieve the delivery status of sent MMS messages. This service uses the client credentials authorization model (see paragraph 1.2 for more details) with the *ServiceConstants.ScopeType.SEND_MMS* scope.

The MMS service contains the following classes:

- *AttMMSCallbackProcessor* for receiving messages
- *AttMMSOutbox* for sending messages
- *AttAttachment* represents the document that can be attached to HTTP based messages
- *AttMMS* represents the MMS
- *AttMMSOutbox* represents the outgoing messages
- *AttMMSOutboxStatus* represents the MMS sending status
- *AttMessageDeliveryStatus* represents the delivery status
- *AttMMSCallbackProcessor* provides callback functionality to process incoming MMS

3.1. Sending MMS messages

To send an MMS message, perform the following steps:

1. Create an instance of the *AttMMSOutbox* class with an *AttAuthorization* object.

```
AttMMSOutbox box = new AttMMSOutbox(auth);
```

2. Create and initialize the attachments.

```
AttAttachment attachment = new AttAttachment(fileContentBlob,
'file.txt', 'text/txt');
```

3. Create an instance of the *AttMMS* class.

```
AttMMS message = new AttMMS();
```

4. Set the properties of the *AttMMS* object.

```
message.phoneNumber = '4258028620';
message.subject = 'test';
message.attachments.add(attachment);
```

5. Pass the MMS object to the *sendMessage* method of *AttMMSOutbox* object and save the returned status.

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

```
AttMMSOutboxStatus sendStatus = box.sendMessage(message);
```

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

3.2. Handling MMS Sending and MMS Delivery Data

NOTE: Don't confuse sending and delivery information. In this context, *sending* means sending MMS from the application to AT&T servers, while *delivery* means transferring MMS from AT&T servers to the recipient mobile device.

When an MMS message is sent, the *sendMessage* method returns an *AttMMSOutboxStatus* object that contains the following information:

- *ID* is the unique identifier for the MMS message
- *resourceUrl* is the URL than contains the MMS message delivery information

NOTE: Use *getDeliveryStatus* to get the MMS message delivery information. It performs a call to AT&T API and returns the delivery status as an *AttMessageDeliveryStatus* object, which contains the *deliveryInfoList* property with a list of *DeliveryInfo* objects.

To get MMS message delivery information, perform the following steps:

1. Call the *getDeliveryStatus* method.

```
AttMessageDeliveryStatus deliveryStatus =
sendStatus.getDeliveryStatus();
```

2. Use the returned *AttMessageDeliveryStatus* object to get delivery information.

```
for (DeliveryInfo deliveryInfo : deliveryStatus.deliveryInfoList) {
      System.debug(deliveryInfo.id);
      System.debug(deliveryInfo.address);
      System.debug(deliveryInfo.deliveryStatus);
}
```

3.3. Saving and Restoring MMS Sending Statuses to and from Custom Objects

Storing MMS sending statuses into custom objects is required in some cases. For example, you need to send an MMS, provide the list of sent MMS messages, and allow the user to check the statuses of the messages. All the information except the MMS message ID can be restored by calling the *restoreOutboxStatusByMMSId* method in the *AttMMSOutbox* class. Make sure that you continue to use the same short code for MMS. Otherwise, you won't be able to get the delivery status.

To save the MMS message ID:

1. Get the *AttMMSOutboxStatus* object returned by the *sendMessage* method and save the ID property.

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

```
String mmsId = sendStatus.id;
```

2. Save the ID.

To restore an MMS message from a stored ID, perform the following steps:

1. Load the ID from the custom object.
2. Create a new *AttMMSOutbox object.*

```
AttMMSOutbox box = new AttMMSOutbox(auth);
```

3. Call the *restoreOutboxStatusByMMSId* method with the message ID to restore *AttMMSOutboxStatus*.

```
AttMMSOutboxStatus sendStatus =
box.restoreOutboxStatusByMMSId(mmsForCheckId);
```

4. Use the *getDeliveryStatus* method of the *AttMMSOutboxStatus* object to get the delivery status.

```
AttMessageDeliveryStatus deliveryStatus =
sendStatus.getDeliveryStatus();
```

3.4. Receiving MMS

You can receive MMS messages only by using callbacks.

To receive MMS messages:

1. Create a class that inherits the *AttMMSCallbackProcessor* class.
2. Implement the *processMessage* method, which is called when a message is received.

To process incoming MMS messages, perform the following steps:

1. Inherit the *AttMMSCallbackProcessor* class and override the *processMessage* method. Method implementation may contain custom logic, such as storing the MMS message as or resending it in an email message.

```
public class MMSCallbackProcessor extends AttMMSCallbackProcessor {
      public override void processMessage(AttMMS mms) {
            Messaging.SingleEmailMessage mail = new
Messaging.SingleEmailMessage();
            mail.setToAddresses(new String[]{'foo@example.com'});
            mail.setSenderDisplayName('John Doe');
            mail.setSubject('MMS: ' + mms.subject);
            mail.setPlainTextBody('Attachments count: ' +
mms.attachments.size());
            Messaging.sendEmail(new Messaging.Email[]{mail});
```

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

```
      }
}
```

2. Create the REST callback in your class by defining the *doPost* method. This method should create the instance of the class overriding the *AttMMSCallbackProcessor*.

```
@RestResource(urlMapping='/MMSInboxCallback')
global class DemoMMSInboxCallback {

    @HttpPost
    global static void doPost() {
        try {
            //Call
            new MMSCallbackProcessor();
        } catch (Exception e) {
            //Exception handling
        }
    }
}
```

3. Go to the *SalesForce* configuration page and setup your class (*DemoMMSInboxCallback* in that case) as REST service.
4. Go to the AT&T configuration page and setup the URL of the REST service as *MMS Mobile Originated URI*.

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

4. Speech Service

The Speech service provides authorized applications the ability to convert speech to text using different speech contexts served by different applications in the speech engine. This service uses the client credentials authorization model (see paragraph 1.2 for more details) with the *ServiceConstants.ScopeType.SPEECH* scope*.*

The Speech service contains the following class:

- *AttSpeech,* which represents speech data and additional parameters

4.1. Converting Speech to Text

To convert speech to text, perform the following steps:

1. Create an instance of the *AttSpeech* class with the client credentials authorization object and the *Speech* service scope type.

```
AttSpech speech = new AttSpeech (new
AttClientCredentialsAuthorization(ServiceConstants.ScopeType.SPEECH);
```

2. Set the properties of the *AttSpeech* object.

```
speech.speechContext = AttSpeech.SpeechContext.GENERIC;
speech.fileType = AttSpeech.FileType.WAV;
speech.fileBlob = new Blob(fileContent);
```

3. Call the *convert* method to convert the audio file to text.

```
AttSpeechResult response = speech.convert();
```

The *AttSpeechResult* object contains the following fields:

1. *Recognition*, which is the recognition with the following properties:

- *responseId*, which is the unique identifier for this specific translation
- *nBest*, which is a list of *NBest* objects with the following properties:
  - *wordScores*, which specifies the confidence scores for each of the strings in the words array. Each value ranges from 0.0 (least confident) to 1.0 (most confident), inclusive.
  - *confidence*, which specifies the confidence value of the *Hypothesis*. Each value ranges from 0.0 (least confident) to 1.0 (most confident) inclusive.

2. *Grade*, which contains a machine-readable string indicating an assessment of utterance/result quality and the recommended treatment of the *Hypothesis*. The

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other
AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T
affiliated companies. All other marks contained herein are the property of their respective owners.

assessment reflects a confidence region based on prior experience with similar results. The value *accept* stands for the hypothesis value with the acceptable confidence, the value *confirm* stands for the value that should be confirmed independently due to lower confidence, and the value *reject* stands for the hypothesis value that should be rejected due to low confidence.

3. *ResultText*, which contains a text string prepared according to the output domain of the application package. The string is typically a formatted version of the *Hypothesis*, but the words may have been altered through insertions, deletions, substitutions to make the result more readable or usable for the client.
4. *Words*, which specifies the words of the *Hypothesis* split into separate strings. The value may omit some of the words of the *Hypothesis* string and may be empty, but does not contain words not in *Hypothesis*.
5. *LanguageId*, which specifies the language used to decode *Hypothesis* using the two-letter ISO 639 language code, hyphen, two-letter ISO 3166 country code in lower case, such as *en-us*. All contexts support only English except for Generic, which supports Spanish as well.
6. *Hypothesis*, which contains the transcription of the audio.

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other
AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T
affiliated companies. All other marks contained herein are the property of their respective owners.

5. WAP Push Service

The WAP Push service enables authorized applications to send WAP Push messages from the network to the AT&T mobile device. This service uses the client credentials authorization model with the *ServiceConstants.ScopeType.WAP_PUSH* scope (see paragraph 1.2 for more details).

The WAP Push service contains the following classes.

- *AttWAPPush***,** which is the base class for the WAP Push service providing methods to send messages and retrieve results from the WAP Push service
- *AttWAPPushResult***,** which contains the result of sending a WAP Push message

5.1. Sending WAP Push Messages

To send a WAP Push message, perform the following steps:

1. Create an instance of the *AttWAPPush* class with the client credentials authorization object and the WAP Push service scope type.

```
AttWAPPush wapPush = new AttWAPPush(new
AttClientCredentialsAuthorization(ServiceConstants.ScopeType.WAPPUSH));
```

2. Set the WAP Push properties and send the message.

```
wapPush.phoneNumbers = new Set<String> {
 'tel:6508631130',
 'tel:6502965430'
 };
wapPush.url = 'http://developer.att.com';
wapPush.message = 'Test WAP Push message';
AttWAPPushResult result = wapPush.sendWAPPush();
```

5.2. Handling WAP Push Message Results

The *sendWAPPush* method returns an AttWAPPushResult object with the following property:

- *id*, which specifies the unique identifier for the message

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

6. Device Capabilities Service

The Device Capabilities service enables authorized applications to get details from the network for a particular mobile terminal. This service uses Consent Authorization Workflow model (see paragraph 1.1 for more details) with *ServiceConstants*.ScopeType.*DEVICE*_CAPABILITIES scope.

The Device Capabilities service contains the following classes:

- *AttDeviceCapabilitiesService*, which is the base class for the Device Capabilities service providing a method to get device capabilities
- *AttDeviceCapabilities*, which represents device information
- AttDeviceCapabilities.*DeviceInfo*, which represents information about the device
- AttDeviceCapabilities.*DeviceId*, which represents the IMEI of the device
- AttDeviceCapabilities.Capabilities, which represents information about the device

In order to make the request, the following conditions must apply:

1. The request can only be made from a mobile browser.
2. The mobile device must use AT&T internet connection.
3. The End User Authorization is required for the service.

6.1. Receiving Device Capabilities

To get the device capabilities:

1. Create an instance of the *AttDeviceCapabilitiesService* class with an *AttAuthorization* object.

```
AttDeviceCapabilitiesService dCService = new
AttDeviceCapabilitiesService(auth);
```

2. Call the *getDeviceCapabilities* method.

NOTE: The returned *AttDeviceCapabilities* object contains the device capabilities.

```
AttDeviceCapabilities deviceCapabilities;
deviceCapabilities = dCService. getDeviceCapabilities();
```

The *AttDeviceCapabilities* object contains the *deviceInfo* property, which is a *DeviceInfo* object with the following properties:

- *deviceId*, which is a DeviceId object with the following properties:

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

    - *TypeAllocationCode*, which specifies the first 8 digits of the International Mobile Equipment Identity of the device

- *capabilities*, which is a *Capabilities* object with the following properties:

    - *Name*, which specifies the abbreviated code used by AT&T for the mobile device manufacturer and model number
    - *Vendor*, which specifies the abbreviated code used by AT&T for the manufacturer of the mobile device
    - *Model*, which specifies the model number used by AT&T for the mobile device.
    - *FirmwareVersion*, which specifies the firmware release number used by AT&T for the mobile device
    - *UaProf*, which specifies the URL to the website of the device manufacturer where the capability details for the mobile device can be found
    - *MmsCapable*, which specifies the MMS capability of the device
    - *AssistedGps*, which specifies the capability of GPS assistance
    - *LocationTechnology*, which specifies the location technology network supported by the device
    - *DeviceBrowser*, which specifies the name of the browser resident on the device
    - *WapPushCapable*, which specifies the capability of WAP Push

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

7. Location Service

The Location service enables authorized applications to query the location of AT&T Phone Numbers for latitude and longitude using approximate accuracy parameters. This service uses code authorization model with *ServiceConstants.ScopeType.TL* scope (see paragraph 1.3 for more details).

The Location service contains the following classes:

1. *AttLocation* is the base class for Location service and provides methods for invoking and retrieving results from AT&T Location services.
2. *AttLocationResult* is the base interface for location data and provides common methods for retrieving location data.
3. *AttCurrentLocation* stores common information about location.

NOTE: Each type of location result contains current location data.

4. *AttDeviceLocationResult* specifies the device location method result.
5. *AttTerminalLocationResult* specifies the terminal location method result.

7.1. Getting Location Information

To get device location information, perform the following steps:

1. Create an instance of the *AttLocation* class with a code authorization object (see paragraph 1.2 for more details).

```
AttCodeAuthorization      auth      =      new     AttCodeAuthorization();
//              obtain                  end-user                 authorization
...
auth.setAuthorizationCode('AtDQkEf3G8gojYFJMTLr8lwFQ');
AttLocation locationService = new AttLocation(auth);
```

2. Call the *getLocation* method with the *AttLocation.ServiceType.DEVICE_LOCATION* service type.

```
AttLocationResult locationResult =
locationService.getLocation(AttLocation.ServiceType.DEVICE_LOCATION,
1000, AttLocation.Tolerance.DelayTolerant, 3000);
```

To get terminal location information, perform the following steps:

1. Create an instance of the *AttLocation* class with the code authorization object (see paragraph 1.2 for more details).

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

2. Call the *getLocation* method with the *AttLocation.ServiceType.TERMINAL_LOCATION* service type and the created code authorization object.

```
AttLocation locationResult =
locationService.getLocation(AttLocationService.ServiceType.TERMINAL_LOC
ATION, 2000, AttLocationService.Tolerance.LowDelay, 5000, auth);
```

7.2. Handling the Location Service Result

The *getLocation* method returns an *AttLocationResult* object for both location methods. The *AttLocationResult* class contains information about the device location which can be retrieved as an *AttCurrentLocation* object.

```
AttCurrentLocation currLocation =  locationResult.getCurrentLocation();
```

Location information can also be retrieved as a string:

```
String locatonAsString =  locationResult.getCurrentLocationAsString();
```

The *AttCurrentLocation* class contains the following properties:

- *accuracy*, which specifies the accuracy of target MSISDN that is used in the Location request
- *latitude*, which specifies the current latitude of the geographical position for the mobile device
- *longitude*, which specifies the current longitude of the geographical position for the mobile device
- *timestamp*, which specifies the timestamp for the location data

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

8. Payment

The Payment service provides developers and merchants with the ability to directly charge for digital services to an AT&T subscriber bill. A merchant can create new transactions and subscriptions, request the status of the transaction or subscription, and authorize refunds both for transactions and subscriptions. This service uses the client credentials authorization model (see paragraph 1.2 for more details) with the ServiceConstants.ScopeType.PAYMENT scope.

The Payment service contains the following classes:

- *AttPayload,* which initiates a Single Payment payload
- *AttSinglePaymentTransaction*, which represents a Single Payment transaction
- *AttTransactionStatus*, which contains Single Payment transaction status information
- *AttTransactionRefundStatus*, which contains Single Payment transaction refund information
- *AttSubscriptionPayload*, which initiates a Subscription Payment payload
- *AttSubscriptionPaymentTransaction,* which represents a Subscription Payment transaction
- *AttSubscriptionStatus*, which contains Subscription Payment transaction status information
- *AttSubscriptionDetailsStatus*, which contains Subscription Payment transaction details information
- *AttPaymentNotification*, which represents base class for notification service, provides methods for invoking and retrieving results from AT&T notification services
- *AttPaymentNotificationDetails*, which contains information about notification details
- *AttNotificationAcknowledgeResponse*, which contains information about notification acknowledge response
- *AttNotificationCallbackProcessor*, which provides callback functionality to process incoming notification IDs

8.1. Creating Payment Transactions

To create a payment transaction, perform the following steps:

1. Create an instance of the *AttPayload* class.

```
AttPayload  transactionPayload = new AttPayload();
```

2. Set the properties of the *AttPayload* object.

```
transactionPayload.Amount = 2.99;
transactionPayload.Category = 1;
```

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

```
transactionPayload.Description = 'Chess game';
transactionPayload.MerchantTransactionId = 'T20120619105000082';
transactionPayload.MerchantPaymentRedirectUrl =
'https://c.na9.visual.force.com/apex/att_payment_page';
transactionPayload.MerchantProductId = 'Game001';
```

3. Create an instance of the *AttSinglePaymentTransaction* class with the *AttPayload* object and an *AttAuthorization* object.

```
AttSinglePaymentTransaction sPaymentTransaction = new
 AttSinglePaymentTransaction(transactionPayload, auth);
```

4. Call the *start* method and save the returned *PageReference* object (see PageReference on the Salesforce web site for more details).

```
PageReference pageRef = singlePaymentTransaction.start();
```

5. Redirect the user to the *PageReference* to authorize the AT&T subscriber as a buyer.

6. After authorization success or failure, the system redirects to the predefined *transactionPayload.MerchantPaymentRedirectUrl* URL.

7. In the redirected page, handle the URL for Authorization Code of a successful transaction. Otherwise, you will get an error returned by the AT&T server.

```
String transactionAuthCode =
AttSinglePaymentTransaction.handleRedirectUrlForAuthorizationCode();
```

8.2. Getting the Status of Transactions

To get the status of a transaction by the Authorization Code, perform the following steps:

1. Call the *getStatusByAuthorizationCode* method for the *AttSinglePaymentTransaction* object with the Authorization Code returned after a successful payment. These parameters are necessary to get the status of the transaction.

2. Save the *AttTransactionStatus* object returned from the call to the *getStatusByAuthorizationCode* method.

```
AttTransactionStatus transactionStatus = sPaymentTransaction.
getStatusByAuthorizationCode(transactionAuthCode);
```

The *AttTransactionStatus* object contains the following properties:

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

- *Channel*, which represents the merchant channel
- *Description*, which contains a short description of the product that was sent in the original charge request
- *TransactionCurrency*, which represents the transaction currency
- *TransactionType*, which is a *TransactionType* object that specifies the type of the transaction as SUBSCRIPTION or SINGLEPAY
- *TransactionStatus*, which is a *TransactionStatus* object that specifies the current status of the transaction as STATUS_NEW, DECLINED, SUCCESSFUL, FAILED, UNRESOLVED, or CANCELLED
- *ConsumerId*, which specifies the unique user ID generated in the payment system and representing a unique subscriber
- *MerchantTransactionId*, which specifies the original transaction ID generated and sent by the merchant application in the original charge request
- *MerchantApplicationId*, which specifies the merchant application ID from the AT&T payment system indicating the payment transaction
- *TransactionId*, which specifies the unique ID generated for the original charge request
- *ContentCategory*, which specifies the content category sent in the original charge request
- *MerchantProductId*, which indicates the product ID from the merchant sent in the original charge request
- *MerchantId*, which specifies the merchant identifier from the AT&T payment system indicating the payment transaction
- *Amount*, which specifies the amount of the transaction
- *Version*, which specifies the version of the payment service
- *IsSuccess*, which indicates the status of the request
- *OriginalTransactionId*, which represents the original transaction ID
- *IsAutoCommitted*, which indicates whether the transaction was auto-committed

To get the transaction status by the Merchant Transaction ID, perform the following steps:

1. You must have a paid transaction.

2. Call the *getStatusByMerchantTransactionId* method to get the status of the payment transaction.

3. Save the *AttTransactionStatus* object that is returned by *getStatusByMerchantTransactionId*.

```
AttTransactionStatus transactionStatus =
sPaymentTransaction.getStatusByMerchantTransactionId();
```

To get the status of a transaction by the Transaction ID, perform the following steps:

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

1. You need to have a paid transaction and check the transaction status by the Merchant Transaction ID or Authorization Code.

```
sPaymentTransaction.getStatusByMerchantTransactionId();
```

2. Call the *getStatusByTransactionId* method to get the status of the payment transaction.

```
AttTransactionStatus transactionStatus =
sPaymentTransaction.getStatusByTransactionId();
```

Or if you already have the *transactionId* of the given transaction:

1. Call the *getStatusByTransactionId* method with the *TransactionId* to get the status of the payment transaction.

```
AttTransactionStatus transactionStatus =
sPaymentTransaction.getStatusByTransactionId(transactionId);
```

8.3. Refunding Transactions

To refund a transaction, perform the following steps:

1. Get the status for the given transaction as described below to obtain the transaction ID.

2. Create a string containing the reason for the refund.

3. Call the *refund* method with the *refundReason* to refund the given transaction.

```
sPaymentTransaction.getStatusByMerchantTransactionId();
String refundReason = 'Customer was not happy';
AttTransactionRefundStatus transactionRefundStatus =
sPaymentTransaction.refund(refundReason);
```

Or if you already have the *AttTransactionStatus* instance of the given transaction:

1. Create a string containing the reason for the refund.

2. Call the *refund* method with the *TransactionId* field from the *transactionStatus* instance and *refundReason* parameter to get the status of the payment transaction.

```
String transactionId = transactionStatus.TransactionId;
String refundReason = 'Customer was not happy';
AttTransactionRefundStatus transactionRefundStatus =
sPaymentTransaction.refund(transactionId, refundReason);
```

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

The *AttTransactionRefundStatus* object contains the following properties:

- *IsSuccess*, which indicates the status of the request
- *Version*, which specifies the version of the payment service
- *TransactionId*, which specifies a unique identifier for the transaction refund
- *TransactionStatus*, which indicates the current status of the transaction
- *OriginalPurchaseAmount*, which indicates the amount of the original purchase
- *CommitConfirmationId*, which specifies a unique identifier for the refund commitment

8.4. Creating New Subscriptions

To initiate a new subscription:

1. Create an instance of the *AttSubscriptionPayload* class.

```
AttSubscriptionPayload subscriptionPayload = new
AttSubscriptionPayload();
```

2. Set the properties of the *AttSubscriptionPayload* object.

```
subscriptionPayload.Amount = 2.99;
subscriptionPayload.Category = 1;
subscriptionPayload.Description = 'Crossword Monthly';
subscriptionPayload.MerchantTransactionId = 'T20120301211038765';
subscriptionPayload.MerchantPaymentRedirectUrl = '
https://c.na9.visual.force.com/apex/att_payment_page';
subscriptionPayload.MerchantProductId = 'P20120301211038765';
subscriptionPayload.MerchantSubscriptionIdList = ' MSIL1211038765';
subscriptionPayload.IsPurchaseOnNoActiveSubscription = false;
subscriptionPayload.SubscriptionRecurrences = 99999;
subscriptionPayload.SubscriptionPeriodAmount = 1;
```

3. Create an instance of the *AttSubscriptionPaymentTransaction* class with the *AttSubscriptionPayload* and *AttAuthorization* objects.

```
AttSubscriptionPaymentTransaction subscriptionTransaction = new
AttSubscriptionPaymentTransaction(subscriptionPayload, auth);
```

4. Call the *start* method and save the returned *PageReference* object (see PageReference on the Salesforce web site for more details).

```
PageReference pageRef =  subscriptionTransaction.start();
```

5. Redirect the user to the value of the *pageRef* property in the *PageReference* object to authorize the AT&T subscriber.

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

6. After authorization success or failure, the system redirects to the value of the *AttSubscriptionPayload.MerchantPaymentRedirectUrl* property.

7. In the redirected page, you can handle the URL for Authorization Code of a successful subscription or get an error returned by the AT&T server.

```
String subscriptionAuthCode = AttSubscriptionPaymentTransaction.
handleRedirectUrlForSubscriptionAuthorizationCode();
```

8.5. Getting the Status of Subscriptions

To get a subscription transaction status by the subscription Authorization Code, perform the following steps:

1. Call the *getStatusByAuthorizationCode* method for the *AttSubscriptionPaymentTransaction* object with the subscription Authorization Code returned after the successful payment to get the status of the subscription transaction.

2. Save the *AttSubscriptionStatus* object returned by *getStatusByAuthorizationCode*.

```
AttSubscriptionStatus subscriptionStatus = subscriptionTransaction.
getStatusByAuthorizationCode(subscriptionAuthCode);
```

The *AttSubscriptionStatus* object contains the following properties:

- *Channel*, which specifies the merchant channel
- *Description*, which specifies a short description of the product that was sent in the original charge request
- *SubsciptionCurrency*, which specifies the subscription currency
- *SubscriptionType*, which specifies the type of the transaction as SUBSCRIPTION or SINGLEPAY
- *SubscriptionStatus*, which indicates the current status of the subscription
- *ConsumerId*, which specifies the unique user ID generated in the payment system and represents a unique subscriber
- *MerchantTransactionId*, which specifies the original subscription ID generated and sent by the merchant application in the original charge request
- *MerchantApplicationId*, which specifies the merchant application ID from the AT&T payment system indicating the payment transaction
- *SubscriptionId*, which specifies the unique ID generated for the original charge request
- *ContentCategory*, which specifies the content category sent in the original charge request
- *MerchantProductId*, which specifies the product ID from the merchant sent in the original charge request

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

- *MerchantId*, which specifies the merchant identifier from the AT&T payment system indicating the payment transaction
- *Amount*, which specifies the amount of the transaction
- *Version*, which specifies the version of the payment service
- *IsSuccess*, which indicates the status of the request
- *OriginalTransactionId*, which specifies the original transaction ID
- *IsAutoCommitted*, which indicates whether the transaction was auto-committed.
- *MerchantSubscriptionId*, which specifies the original subscription ID to which this subscription is associated
- *SubscriptionPeriodAmount*, which specifies the number of periods that pass between subscription renewals
- *SubscriptionRecurrences*, which specifies the number of times the subscription is renewed when the subscription is created
- *SubscriptionPeriod*, which specifies the subscription period that was sent in the original charge request

To get the subscription transaction status by the Merchant Subscription ID, perform the following steps:

1. You must have a paid subscription.

2. Call the *getStatusByMerchantTransactionId* method with the *AttAuthorization* object to get the status of the payment transaction.

3. Save the *AttSubscriptionStatus* object returned by *getStatusByMerchantTransactionId*.

```
AttSubscriptionStatus subscriptionStatus =
subscriptionTransaction.getStatusByMerchantTransactionId();
```

To get a subscription transaction status by the Subscription ID, perform the following steps:

1. You must have a paid subscription and check the status of the subscription transaction by the Merchant Transaction ID or Authorization Code.

```
subscriptionTransaction.getStatusByMerchantTransactionId();
```

2. Call the *getStatusBySubscriptionId* method to get the status of the payment transaction.

```
AttSubscriptionStatus subscriptionStatus = subscriptionTransaction.
getStatusBySubscriptionId();
```

Or if you already have the *AttSubscriptionStatus* instance of the given transaction:

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

1. Call the *getStatusBySubscriptionId* method with the *SubscriptionId* field from the *subscriptionStatus* instance to get the status of the payment transaction.

```
String subscriptionId = subscriptionStatus.SubscriptionId;
AttSubscriptionStatus subscriptionStatus = subscriptionTransaction.
getStatusBySubscriptionId(subscriptionId);
```

8.6. Getting the Details of Subscription Transactions

To get the detailed information of a subscription transaction, perform the following steps:

1. You must have a paid subscription and check the status of the subscription.

```
subscriptionTransaction.getStatusByMerchantTransactionId();
```

2. Call the *getDetails* method with the *AttAuthorization* object to get the details of the subscription transaction.

```
AttSubscriptionDetailsStatus subscriptionDetails =
subscriptionTransaction. getDetails();
```

If you already have an *AttSubscriptionStatus* object of the transaction, you can get the details of the subscription transaction by performing the following step:

1. Call the *getDetails* method with the value of the *ConsumerId* parameter from the *AttSubscriptionStatus* object to get the details of the subscription transaction.

```
String consumerId = subscriptionStatus.ConsumerId;
AttSubscriptionDetailsStatus subscriptionDetails =
subscriptionTransaction. getDetails(consumerId);
```

The *AttSubscriptionDetails* object contains the following properties:

- *SubsciptionCurrency*, which specifies the subscription currency
- *Status*, which indicates the status used for subscription consumption
- *CreationDate*, which specifies the date, in UTC format, when the subscription is created
- *GrossAmount*, which specifies the subscription renewal charge
- *SubscriptionRecurrences,* which specifies the number of times the subscription is renewed when the subscription is created
- *IsActiveSubscription*, which indicates whether the subscription is active and can be consumed
- *CurrentStartDate*, which specifies the date, in UTC format, when the current subscription period started
- *CurrentEndDate*, which specifies the date, in UTC format, when the current subscription period ends

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

- *SubscriptionRemaining*, which specifies the remaining number of renewals
- *Version*, which specifies the version of the payment service
- *IsSuccess*, which indicates the status of the request

8.7. Refunding Subscriptions

To refund the subscription transaction:

1. Get the status for the given transaction as described below to obtain the subscription ID.

2. Create a string containing the reason for the refund.

3. Call the *refund* method with the *refundReason* parameter to refund the given subscription transaction.

```
subscriptionTransaction.getStatusByMerchantTransactionId();
String refundReason = 'Customer was not happy';
AttTransactionRefundStatus transactionRefundStatus =
subscriptionTransaction.refund(refundReason, auth);
```

Or if you already have the *AttSubscriptionStatus* instance of the given transaction:

1. Call the *refund* method with the subscription ID and the refund reason to get the status of the payment transaction.

```
String subscriptionId = transactionStatus.SubscriptionId;
String refundReason = 'Customer was not happy';
AttTransactionRefundStatus transactionRefundStatus =
sPaymentTransaction.refund(subscriptionId, refundReason);
```

The *AttTransactionRefundStatus* object contains the following properties:

- *IsSuccess*, which specifies the status of the request
- *Version*, which specifies the version of the payment service
- *TransactionID*, which specifies the transaction ID for the refund
- *TransactionStatus*, which specifies the current status of the transaction
- *OriginalPurchaseAmount*, which specifies the amount of the original purchase
- *CommitConfirmationId*, which specifies the unique identifier for the commitment

8.8. Receiving Notification IDs by Callback

To receive a notification ID used for getting notification details and acknowledging notifications, use the *AttNotificationCallbackProcessor* class, which provides callback functionality to process incoming notification IDs. It is also used to parse the notification request data and encapsulate them in the *List<String>notificationIdList* object. The class contains the abstract method *processNotification* that is called when a message is

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other
AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T
affiliated companies. All other marks contained herein are the property of their respective owners.

received and should be implemented in the class inheriting from the *AttNotificationCallbackProcessor*.

To process the incoming notification ID, perform the following steps:

1. Inherit the *AttNotificationCallbackProcessor* class and override the *processNotification(List<String> notificationIdList)* method. The method implementation may contain any custom logic, like persisting IDs as objects and getting notification details.

```
public class NotificationCallbackProcessor extends
AttNotificationCallbackProcessor {
public override void processNotification (List<String>
notificationIdList){
        for (String id : notificationIdList) {
            AttNotificationIdRecord__c notification = new
AttNotificationIdRecord__c();
            notification.NotificationId__c = id;
            insert notification;
        }
    }
}
```

2. Create a REST callback in your class by defining the *doPost* method. This method should create the instance of the class overriding the *AttNotificationCallbackProcessor*.

```
@RestResource(urlMapping='/NotificationCallback)
global class DemoPaymentNotificationCallback{

    @HttpPost
    global static void doPost() {
        try {
            //Call
            new NotificationCallbackProcessor();
        } catch (Exception e) {
            //Exception handling
        }
    }
}
```

3. Go to the *SalesForce* configuration page and setup *DemoPaymentNotificationCallback* as REST service.

4. Go to the AT&T merchant details configuration page and setup the URL of the REST service as *Notification URL*.

8.9. Getting Notification Details

To get the notification details, perform the following steps:

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other
AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T
affiliated companies. All other marks contained herein are the property of their respective owners.

1. You must have a notification ID received by notification callback.

2. Create an instance of the *AttPaymentNotification* class with the notification ID and an *AttAuthorization* object.

```
AttPaymentNotification notification = new
AttPaymentNotification(recievedId, auth);
```

3. Call the *getNotification* method to retrieve the *AttPaymentNotificationDetails* instance.

```
AttPaymentNotificationDetails details = notification.getNotification();
```

The *AttPaymentNotificationDetails* object contains the following properties:

- *getNotificationResponse*, the *AttPaymentNotificationDetails*. *GetNotificationResponse* object containing the summary of the notification
- *Version*, which specifies the version of the payment service
- *IsSuccess*, which indicates the status of the request

The *AttPaymentNotificationDetails.GetNotificationResponse* object contains the following properties:

- *NotificationType*, which specifies the notification type
- *TransactionDate*, which specifies the date and time that the refund transaction occurred
- *OriginalTransactionId*, which specifies the transaction ID of the original transaction
- *ConsumerId*, which specifies the ID associated with the transaction
- *RefundPriceAmount*, which specifies the amount refunded
- *RefundCurrency*, which specifies the currency of the refund
- *RefundReasonId*, which specifies the reason code for the refund
- *RefundFreeText*, which specifies the text description of the reason for the refund
- *RefundTransactionId*, which specifies the transaction identifier of the refunded transaction
- *MerchantSubscriptionId*, which specifies the merchant subscription ID specified in the New Subscription method
- *OriginalPriceAmount*, which specifies the price of the original transaction
- *CurrentPeriodStartDate*, which specifies the start date for the current subscription period
- *CurrentPeriodEndDate*, which specifies the end date for the current subscription period
- *SubscriptionPeriodAmount*, which specifies the number of remaining subscription periods
- *SubscriptionPeriod*, which specifies the time unit for the subscription period

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.

- *SubscriptionRecurrences*, which specifies the number of renewals for subscription
- *SubscriptionRemaining*, which specifies the remaining number of renewals for the subscription

8.10. Acknowledging Notifications

To acknowledge the notification, perform the following step:

1. Call the *acknowledge* method for *AttPaymentNotification* instance.

```
AttNotificationAcknowledgeResponse acknowDetails = notification.acknowledge();
```

The *AttNotificationAcknowledgeResponse* object contains the following properties:

- *Version*, which specifies the version of the payment service
- *IsSuccess*, which indicates the status of the request

© 2012 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. All other marks contained herein are the property of their respective owners.