

Implementing an NDA Signing iPad Application Using Salesforce.com and the DocuSign API

Problem

Your visitors must sign an NDA at your front desk. You want to streamline this process, make your visitor's first impression of your company memorable, and capture information about your visitor in Salesforce.com.

Solution

Use the DocuSign API, VisualForce pages, and a Salesforce Custom Object to have your visitor sign in, and sign an NDA document using an iPad!

Before getting started, you need to get a free Salesforce.com developer account at <https://developer.force.com>, and a free DocuSign developer account at <http://www.docusign.com/devcenter>.

1. Start out by adding DocuSign web services to your authorized endpoints for your Salesforce.com developer account. To do this, go to Setup->Security Controls->Remote Site Settings. Click on the New Remote Site button. Type in "DocuSignDemo" for the Remote Site Name and "https://demo.docusign.net" for the Remote Site URL and make sure the Active check box is checked. Click on Save. This URL will be your endpoint for DocuSign web service calls.
2. Next, create a custom object. This object will be used to store your visitor's information. Start by going to Setup->Create->Objects. Click on the New Custom Object button. Fill in the Label textbox with "NDA Signer", Plural Label: NDA Signers, check Starts with vowel sound, Object Name: "NDA_Signer", Description: "Contains data for an NDA Signer", Record Name: "NDA Signer Name", Data Type: Text. Leave everything else at default and click Save. You'll see a detail of the NDA_Signer custom object with the following at the top:

Custom Object Definition Detail				Edit	Delete
Singular Label	NDA Signer	Description	Contains data for an NDA Signer		
Plural Label	NDA Signers	Enable Reports	<input type="checkbox"/>		
Object Name	NDA_Signer	Track Activities	<input type="checkbox"/>		
API Name	NDA_Signer__c	Track Field History	<input type="checkbox"/>		
		Deployment Status	Deployed		
		Help Settings	Standard salesforce.com Help Window		

3. Now, you need to create the Custom Fields & Relationships.

- a. Click on the New button.

Step 1: Select the "Text" radio button, then click Next.

Step 2: Type "here to see" for the Field Label. Type "128" for the Length and make sure the Field Name is "Here_to_see". Leave rest at defaults Click Next.

Step 3: All of the Visible checkboxes should be checked. Click Next.

Step 4: The Add Field checkbox should be checked. Click Save & New.

- b. Create another text field doing the steps above for the Field Label "Selected name" with a Field Name of "Selected_name".
- c. Create another text field doing the steps above for the Field Label "Company" with a Field Name of "Company".
- d. Create a new email field.

Step 1: Email data type. Click Next.

Step 2: Field Label is "Email", Field Name is "Email", leave rest at default. Click Next.

Step 3: accept defaults. Click Next.

Step 4: accept defaults. Click Save and New.

- e. Create a Purpose of Visit picklist field.

Step: 1: Picklist data type. Click Next.

Step 2: Field Label is "Purpose of Visit", list of values is "Business", "Interview", "Personal", Check Use first value as default value, Field Name is "Purpose_of_Visit", leave rest at defaults. Click Next.

Step 3: Check Visible insuring all Field-Level Security items are checked. Click Next.

Step 4: Check NDA_Signer Layout. Click Save and New.

- f. Create a sign-in field

Step: 1: Date/Time data type. Click Next.

Step 2: Field Label is "sign in", Field Name is "sign_in", leave rest at defaults. Click Next.

Step 3: Check Visible insuring all Field-Level Security items are checked. Click Next.

Step 4: Check NDA_Signer Layout. Click Save and New.

- g. Create another Date/Time file doing the steps above for the Field Label "sign out" with a Field Name of "sign_out".

- h. Create signed in? field

Step: 1: Formula data type. Click Next.

Step 2: Field Label is "Signed In?", Field Name is "signed_in", Formula Return Type is Text. Click Next.

Step 3: Paste the following formula into the formula edit box:

```
if(and(isblank(sign_out__c), not(isblank(sign_in__c))), "Yes", "No")
```

Click Next.

Step4: Check Visible insuring all Field-Level Security items are checked.
Click Next.

Step 5: Check NDA_Signer Layout. Click Save and New.

- i. Create a Validate Method field

Step 1: Picklist data type. Click Next.

Step 2: Field Label is "Validate method", list of values is "SHOWID", "PHONE", "RSAID", check Use first value as default value, Field name is "Validate_method". Click Next.

Step 3: Check Visible. Click Next.

Step 4: Check NDA_Signer Layout. Click Save.

The Custom Fields and Relationships area of the NDA_Signer Custom Object detail should look like:

Custom Fields & Relationships				New	Field Dependencies
Action	Field Label	API Name	Data Type		
Edit Del	<u>Company</u>	Company__c	Text(140)		
Edit Del	<u>Email</u>	Email__c	Email		
Edit Del	<u>Here to see</u>	Here_to_see__c	Text(255)		
Edit Del Replace	<u>Purpose of visit</u>	Purpose_of_visit__c	Picklist		
Edit Del	<u>selected_name</u>	selected_name__c	Text Area(255)		
Edit Del	<u>signed in?</u>	signed_in__c	Formula (Text)		
Edit Del	<u>sign in</u>	sign_in__c	Date/Time		
Edit Del	<u>sign out</u>	sign_out__c	Date/Time		
Edit Del Replace	<u>validate_method</u>	validate_method__c	Picklist		

4. Now, we must create the proxy classes used to make DocuSign API calls. There are three classes we must create, all from DocuSign WSDLs. Download the following WSDLs and save them to your desktop:

<https://demo.docusign.net/api/3.0/Schema/dsapi-send.wsdl>

<https://demo.docusign.net/api/3.0/Schema/dsapi-document.wsdl>

<https://demo.docusign.net/api/3.0/Schema/dsapi-account.wsdl>

Now, go to Setup->Develop->Apex Classes and click on the Generate from WSDL button. Browse for the dsapi-send.wsdl.xml file you saved to your desktop and click the Parse WSDL button. Type "DocuSignAPI" for the Apex Class Name instead of the default and click on the Generate Apex code button.

Do the same for dsapi-document.wsdl.xml calling it "DocuSignAPI_document".

Do the same for dsapi-account.wsdl.xml calling it "DocuSignAPI_account".

5. The rest of the code and resources we will be adding are located at DocuSign's SDK site at <https://github.com/docuSign/DocuSign-eSignature-SDK>. Download our SDK by clicking on the Downloads button. Select your favorite compression type, Save it somewhere, and uncompress it.
6. You'll first need to upload the static resource file. Back to your Salesforce page, go to Setup->Develop->Static Resources. Click the New button. Type "ndaStyles" in the Name textbox. Browse for the following file:

`<root_dir_of_sdk_download>/Salesforce/NDAKiosk/staticresource/ndaStyles.zip.`

Select Public for the Cache Control and click Save.

NOTE: The other two files in the staticresources dir is a ndastyles.css file that is the stylesheet that contains the branding styles, and an icon_ipad.png file that tells the iPad Safari browser what icon to use when this application is bookmarked to the iPad Home Page. You can change this, zip it to a new ndaStyles.zip file, and upload it again to change the look of the NDAKiosk application.

7. Now we will create the Apex controller class that is used by the application. Go to Setup->Develop->Apex Classes and click on the New button. From your DocuSign SDK download, open
`<root_dir_of_sdk_download>/Salesforce/NDAKiosk/classes/NDAKioskController.cls`
in a text editor. Copy all of the text and paste it to Apex Class Edit area. Click Quick Save. You will receive an error that says: Error: Page ndadone does not exist. Click the Create Page ndadone link. Click the Quick Save button again and you will receive a new error for the ndapop page. Click the Create Page link again. Repeat these two steps for the ndanamereres, ndanotifyreception, ndasign, ndadonesignout, ndasignout, and ndawelcome pages. Clicking the Quick Save button after creating the ndawelcome page should not result in any errors. Click the Save button.
8. Now you'll add the code for these eight pages. The code is located in the `<root_dir_of_sdk_download>/Salesforce/NDAKiosk/pages` directory in the downloaded DocuSign SDK. In your Salesforce site, go to Setup->Develop->Pages. Click the N link above the list to shorten it. You should see the six pages that were created in step 7. Click the Edit link for each page, and completely replace the code in the VisualForce Markup area with the code in the corresponding .page file (e.g., replace all of the code for the VisualForce ndadone page with the text in the ndadone.page file).
9. We're almost there! You must upload a template to your DocuSign account and get some information that will allow the NDAKioskController class to call your account and create NDA documents from your template. Login to your DocuSign dev account. Select the Templates folder on the left of the Console. Click the

Browse button next to the Upload Template textbox and upload the template file at

<root_dir_of_sdk_download>/Salesforce/NDAKiosk/DocuSign_Visitor_NDA.xml.

Click on the DocuSign Visitor NDA template then click on the Open link above. You will see something like the following at the top of the template:


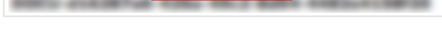


Note the string after the Template ID. This will be need to be pasted into the NDAKioskController class created in step 7. Click the red X in the upper right-hand corner and click No when the Save Template? dialog comes up.

10. Now you will need your DocuSign credentials to paste into the NDAKioskController class. These credentials are used to authenticate your calls to the DocuSign web service. In the DocuSign member console go to Preferences->API. At the top of the page you'll see the GUID for an Integrators Key. If you have not done so, click the Activate button next to the GUID in the Request a new Integrator's Key area. You should now see the following:

API Information

API Access: **Enabled**

Integrator Keys	Description	Production Activation
	DocuSign	Submit for Certification
	DocuSign Reception	Submit for Certification

Request a new Integrator's Key: 

Key Description: DocuSign Reception

Set Key from another DocuSign environment:

Key Description:

Integrator's Key allows you to create software that makes API requests to any account on the DocuSign system. Integrator Keys are required for all API calls to go through a certification session to get the key enabled. You can contact certifications@docuSign.com to schedule this session when your software

API UserName: 

API Password: <your current password>

API Account ID: 

API credentials allow you to make web service calls using unique identifiers or readable e-mails. SOAP Headers are required for API version 3.0.

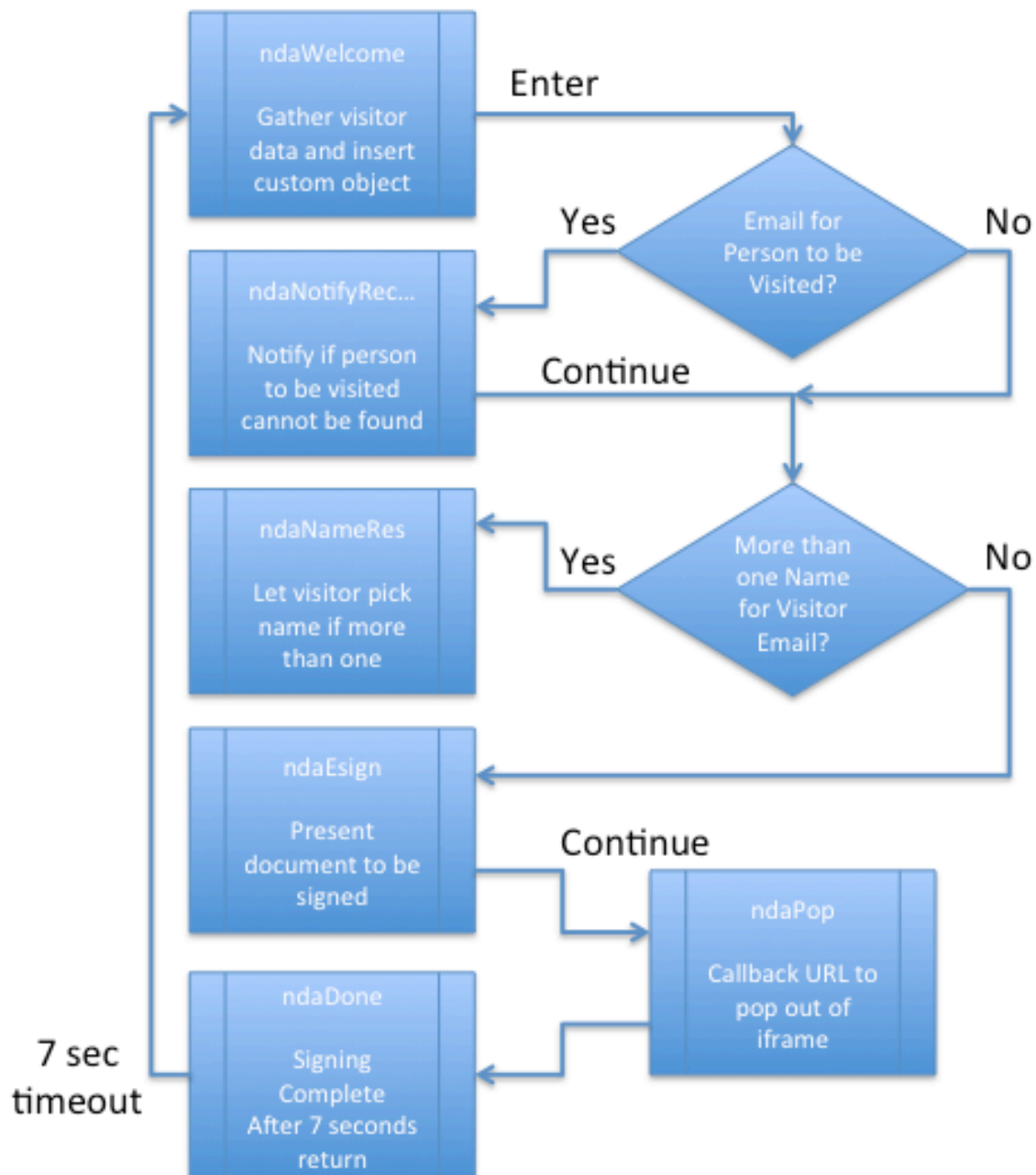
Note the Integrators Key GUID (be sure it's the circled one), the API UserName (either the GUID or your email address will work for the userID string in the NDAKioskController class), and the API Account ID GUID. Go to your Salesforce dev site and select Setup->Develop->Apex Classes->Edit NDAKioskController. Locate the strings starting with TODO in lines 17 through 22. Replace the accountID string with the API Account ID GUID noted above. Replace the userID string with the API UserName above. Replace the password string with the password you used to login to your DocuSign dev member console. Replace the integratorsKey string with the Integrators Key circled above. Replace the templateID string with the Template ID GUID

noted in step 9. Replace the devHost string with the host part of the URL of the page you're currently on (e.g., "https://na3.salesforce.com/", don't forget the trailing "/"). Click Save. You're done!

11. The "Here to see:" input field on the welcome page looks at your Salesforce Contacts to figure out whom to email for notification a visitor has arrived to see them. You should add a few test contacts, and include email addresses you can monitor. Without a valid email address, the "Here to see" feature will present a message saying: "The person you are here to see doesn't seem to be listed. Please notify the receptionist, then press continue".
12. You can now try the application. Type in the URL of the page by adding "apex/ndawelcome" to the your host URL. You will be taken to your Welcome page, and you can try out the application either on your desktop, or your iPad!

Control Flow

The application consists of six pages. The NDAKioskController class determines flow from page to page. The following diagram illustrates the control flow.



Some Code Walkthrough

We will concentrate on the code that is unique to the DocuSign API and how it works. The "SendNDANow" private NDAKioskController class created in step 7 implements most of this functionality. It is this method that builds an "Envelope"

from the data in an NDA_Signer custom object (created in step 2) and a template (uploaded in step 9).

An Envelope is a transaction container that includes documents, recipient information, workflow, and data.

The first thing this method does is to create a "Recipient".

A Recipient is a person who receives an Envelope. This recipient can have one of several roles: signer, carbon copy recipient, editor, agent, or certified delivery.

By default, validation is "trusted" meaning the person who will sign the document is known to be the correct one. There are other levels of validation that DocuSign offers including voice identification by phone, and RSA identification. The following code makes these choices based on what was selected on the welcome page:

```
if (signer.validate_method__c == 'RSAID') {
    recipient.RequireIDLookup = true;
}
else if (signer.validate_method__c == 'PHONE') {
    recipient.RequireIDLookup = true;
    recipient.IDCheckConfigurationName = 'Phone Auth $';
    DocuSignAPI.RecipientPhoneAuthentication phoneAuth = new
    DocuSignAPI.RecipientPhoneAuthentication();
    phoneAuth.RecipMayProvideNumber = true;
    recipient.PhoneAuthentication = phoneAuth;
}
else {
    recipient.RequireIDLookup = false;
}
```

The following code informs the DocuSign web service that we wish to initiate an Embedded signing experience.

An Embedded Signing Experience means that the document to be signed will appear embedded in a web application. A Remote Signing Experience means that the document will be emailed to a recipient for signing.

```
// make recipient captive for embedded experience
recipient.CaptiveInfo = new DocuSignAPI.RecipientCaptiveInfo();
recipient.CaptiveInfo.ClientUserId = '1';
```

Next we want to use a template that is stored in your DocuSign member account (a "server-side template" as the document to be signed:

```
// Create object for the NDA server-side template
DocuSignAPI.TemplateReference ndaTemplate = new
    DocuSignAPI.TemplateReference();
ndaTemplate.Template = templateId;
ndaTemplate.TemplateLocation = 'Server';
```

We have some "tabs" on our template that need to be filled in from the information we entered on the ndawelcome page. Each tab has a label and a value. The following code fills these in:


```
// Add data for fields
DocuSignAPI.TemplateReferenceFieldDataDataValue fd1 = new
    DocuSignAPI.TemplateReferenceFieldDataDataValue();
fd1.TabLabel = 'Full Name 1';
fd1.Value = recipient.UserName;

DocuSignAPI.TemplateReferenceFieldDataDataValue fd2 = new
    DocuSignAPI.TemplateReferenceFieldDataDataValue();
fd2.TabLabel = 'Company 3';
fd2.Value = signer.Company__c;
```

Now we make the call to `CreateEnvelopeFromTemplates`. If we had not created the `CaptiveInfo` object above, the document would have been sent to the Recipient's email. Instead, the method returns an envelope id that can be used to get a unique session limited URL, called a "token URL" that can be used in an `iframe` element for an embedded signing. The code to get this URL is as follows:

```
DocuSignAPI.RequestRecipientTokenClientURLs clientURLs = new
    DocuSignAPI.RequestRecipientTokenClientURLs();

clientURLs.OnAccessCodeFailed = getPopURL() + '?Id=' + signer.id +
    '&event=OnAccessCodeFailed&envelopeid=' + envelopeID;
clientURLs.OnCancel = getPopURL() + '?Id=' + signer.id +
    '&event=OnCancel&envelopeid=' + envelopeID;
clientURLs.OnDecline = getPopURL() + '?Id=' + signer.id +
    '&event=OnDecline&envelopeid=' + envelopeID;
clientURLs.OnException = getPopURL() + '?Id=' + signer.id +
    '&event=OnException&envelopeid=' + envelopeID;
clientURLs.OnFaxPending = getPopURL() + '?Id=' + signer.id +
    '&event=OnFaxPending&envelopeid=' + envelopeID;
clientURLs.OnIdCheckFailed = getPopURL() + '?Id=' + signer.id +
    '&event=OnIdCheckFailed&envelopeid=' + envelopeID;
clientURLs.OnSessionTimeout = getPopURL() + '?Id=' + signer.id +
    '&event=OnSessionTimeout&envelopeid=' + envelopeID;
clientURLs.OnSigningComplete = getPopURL() + '?Id=' + signer.id +
    '&event=OnSigningComplete&envelopeid=' + envelopeID;
clientURLs.OnTTLExpired = getPopURL() + '?Id=' + signer.id +
    '&event=OnTTLExpired&envelopeid=' + envelopeID;
clientURLs.OnViewingComplete = getPopURL() + '?Id=' + signer.id +
    '&event=OnViewingComplete&envelopeid=' + envelopeID;

// assumes apiService = preconfigured api proxy
try {
    token = dsApiSend.RequestRecipientToken(envelopeId,
        recipient.captiveinfo.ClientUserId, recipient.UserName,
        recipient.Email, assert, clientURLs);
} catch (CalloutException e) {
    System.debug('Exception - ' + e);
    errMsg = 'Exception - ' + e;
    return ''; //TODO: send to error landing place
}
return token;
```

Note the series of `RequestRecipientTokenClientURLs`. These are callback URLs that are called by the DocuSign web service depending on the event that ended the document signing. Here we are calling the `ndapop` page we created to "pop" out of the `iframe` and send us to the `ndadone` page.