AVAYA

avaya.com

Avaya Contact Center
Integration to Salesforce
Date: March 2014 Version 1.4

Avaya Contact Center & Salesforce Integration Solutions

TABLE OF CONTENTS

1 - INTRODUCTION AND OVERVIEW

This white paper applies to the applications Avaya Aura® Contact Center from release 6.1 Service Pack 2 and also Avaya Contact Center Select which will be GA June 2014. It will describe how to perform two different types of integration to Salesforce.Com CRM which can also be described as (i) Client side Integration and (ii) Server side Integration.

It is assumed that there is in existance an enterprise Salesforce account which contains the customer records. As part of this application note, we will create such an account in order to show the process. We will then show how you can achieve a client side type integration which is essentially a screen pop of a customer record from Salesforce based on a single parameter calling number and also a screen pop based on multiple parameters calling number and customer number. We will then describe how to achieve a more complex server side integration to perform a lookup of a customer's priority or preferred agent while the contact is queueing in order to decide which agent or skillset will be assigned the contact. Apex code examples are also presented in this white paper.

Client Side Integration

Client side integration is the easiest and simplest integration available out of the box with Avaya Contact Center It enables the screen pop of customer details from Salesforce using Avaya Contact Center "contact intrinsics" using a **sample** screen pop configuration in Avaya Contact Center To create this client side integration solution there are four key pieces of work to be completed:-

- Enabling the Avaya Salesforce application code (often referred to as a "plug-in") within the newly created Salesforce account. This is used to pass information to and from Avaya Contact Center Later on we will see how this is achieved.
- Enabling the agent workstation (client) to securely connect to the Enterprise Salesforce account which contains all of the customer details
- Configuring Avaya Contact Center multimedia component to enable Salesforce screen pop when contact is presented to the agent
- Performing first test call to ensure authorisation completes successfully between client workstation (agent PC) and Salesforce CRM.

Server Side Integration

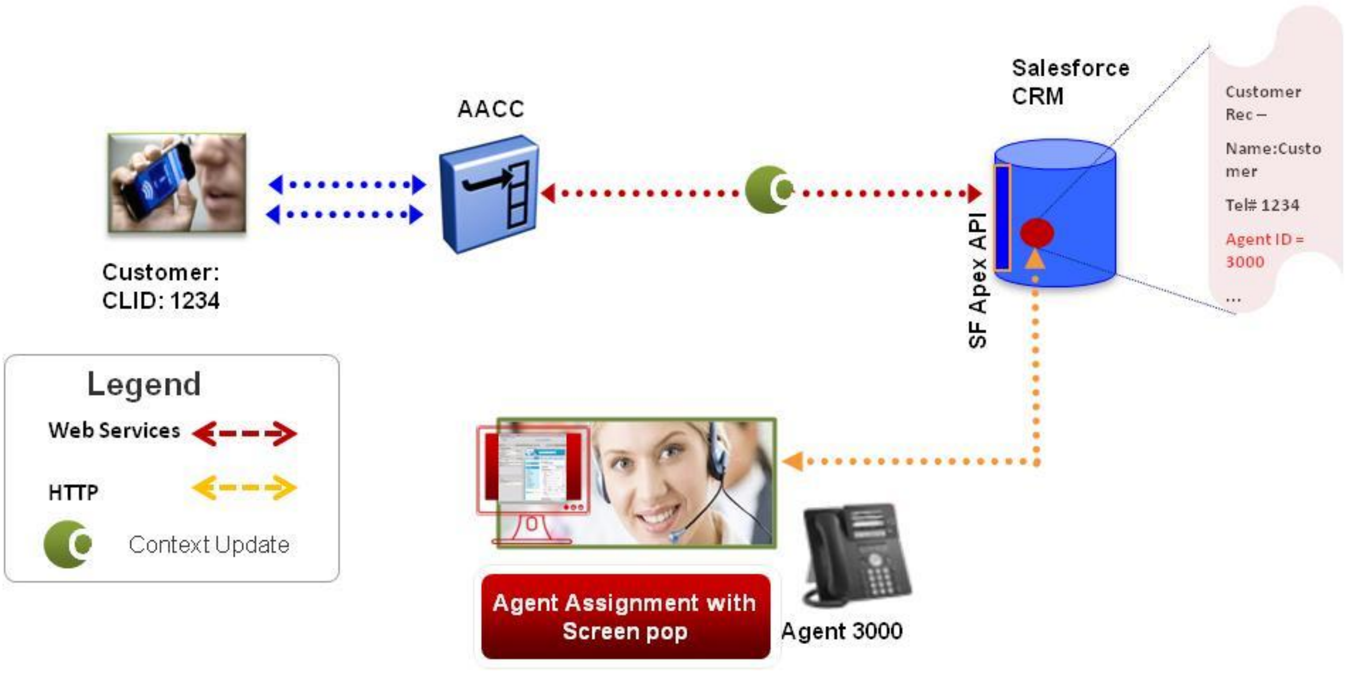
Server side integration is a advanced workflow solution between Avaya Aura Contact Center and Salesforce CRM. Server side integraiton delivers the capability to perform a lookup into Salesforce directly from the Avaya Contact Center workflow in order to make a work assignment/routing decision. For example assign a caller to the preferred agent who handled the caller using the agent ID parameter. Or check the customer priority field in

Salesforce before assigning an agent to the caller. Server side integraiton will utilise the client side configuration described above.

Server Side integration is an optional step while client side integration is required if any screen pops of customer details from Salesforce are required. To create a server side integration there are four key pieces of work to be completed:- (note client side must be completed before server side can be made operational)

- Enabling the CCMS component to securely logon to Salesforce CRM using pre-defined parameters.
- Developing new code modules within the Salesforce API's (known as APEX) which will be used to retrieve the "agent ID" or "Customer Priority" and pass it back to Avaya Contact Center.
- Defining Orchestration Designer (formerly Service Creation Environment) workflow which will assign the customer contact to the preferred agent with the correct agent ID.
- Configuring the screen pop parameters using CCMM administration application – Client side integration.

The methods described in this application note can be applied to all integrations from Avaya Contact Center to Salesforce CRM. The following diagram shows the high level architecture of the Avaya Contact Center – Salesforce concept.



In terms of the contact flow represented by the above illustration the following are the key steps – note the prefix's SS (server side) and CS (client side) are included for completeness.

- Incoming customer contact (voice) arrives into Avaya Contact Center (anchored on Avaya Media Server) – this example is voice but any contact can use this integration including email contacts i.e. pass in email address to Salesforce
- (**SS**) Before the contact is assigned to an agent in a skillset, Avaya Contact Center makes a web service request to Salesforce CRM which includes the customer phone number 1234 (optionally it can be any unique contact information such as email address, web page, SIP URI or customer entered data from self service)
- (**SS**) This request is for the "Preferred Agent" value associated with the customer record in Salesforce. If a match is found in the Salesforce DB, Salesforce returns the Agent ID associated with this phone number. If no match is found, Salesforce can return any other defined value.
- Within Avaya Contact Center, the workflow now utilises the Agent ID (if valid value returned) and performs an "Assign to AgentID" command.
- Incoming contact is now delivered to the agent with corresponding agent ID.
- (**CS**) Using Avaya Contact Center sample screen pop functionality, the customer is record is presented to the agent using Agent Desktop.

The remainder of this application note will now describe the steps required to deliver the above contact assignment and workflow capabilities. Note the document is divided into two parts to reflect the two different integration points.

PART I

2 – Salesforce Account Creation & Integration

In order to integrate with Salesforce, the enterprise must have a Salesforce instance/deployment with the appropriate account(s) configured to hold all customer records. A full description on how Salesforce accounts are created is beyond the scope of this document. Instead we will show for completeness one method to setup the Salesforce Developer account for the purposes of our integration.

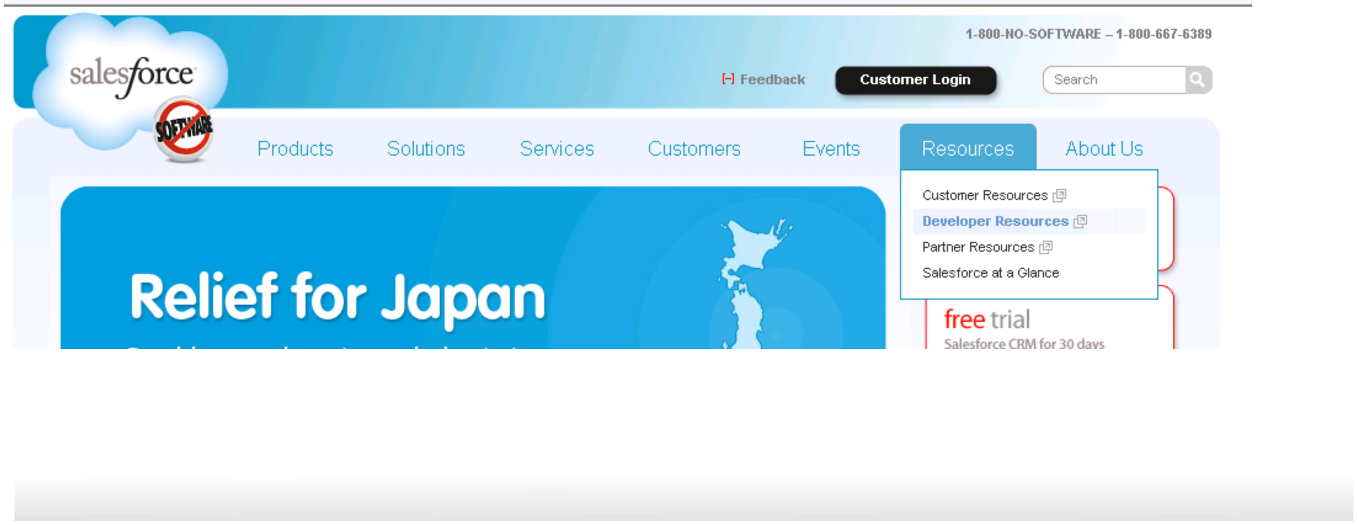
For information to be exchanged between Avaya Contact Center and Salesforce CRM, a secure, authenticated connection must be established between the two systems. The Salesforce CRM platform allows enterprises to register and setup an account(s) specific to the enterprise.

*For integration between Salesforce and Avaya Contact Center you will require a **DEVELOPER** level Salesforce account*. Please remember to specify developer access when registering for Salesforce.

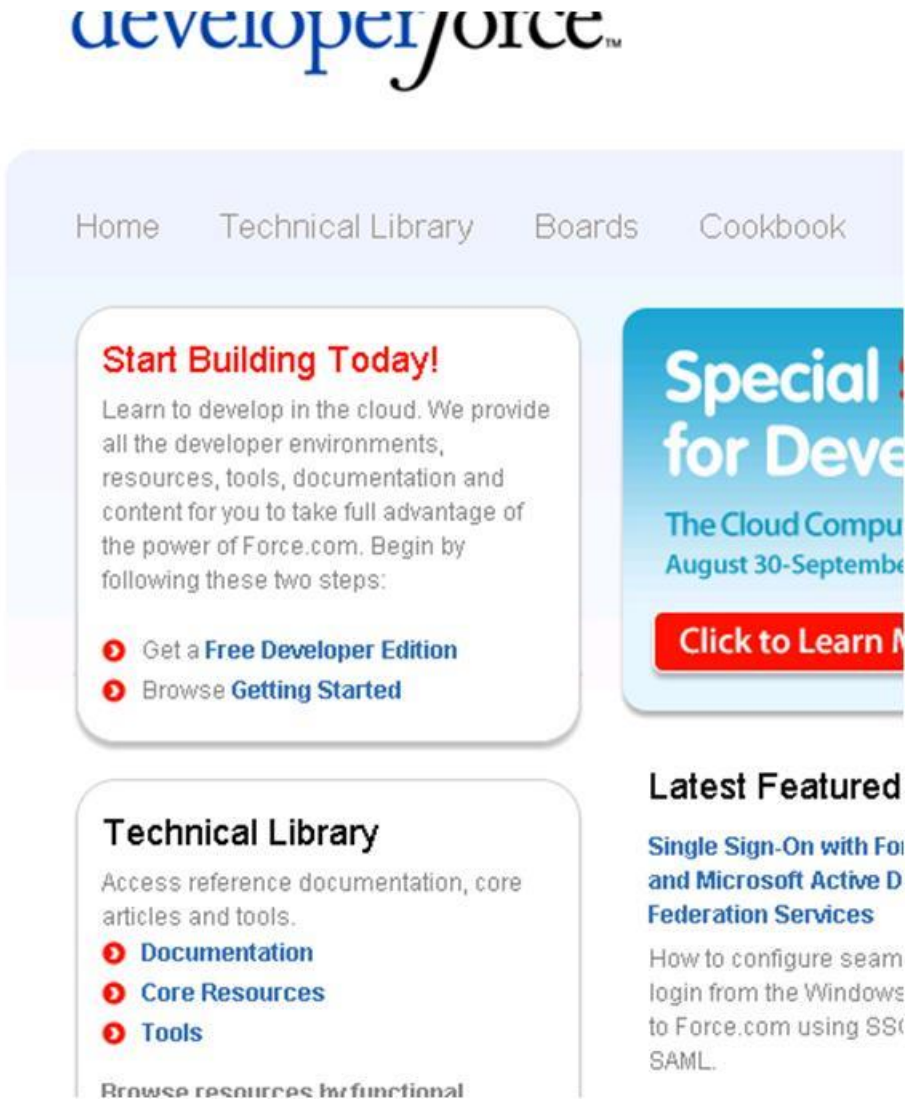
In order to create a Salesforce developer account, you will be required to enter details of an email account which will be used to send authorisation information from Salesforce. It is assumed that this email account is created and available for use.

Salesforce Developer Account Creation

In order to document and test the procedures described in this document we will create a new Salesforce account called Avaya Galway CEC. Browse http://www.salesforce.com/. Click on *Resources->Developer Resources* as shown here.



Now click on Free Developer Edition as shown here.



developerforce™

Home Technical Library Boards Cookbook

Start Building Today!

Learn to develop in the cloud. We provide all the developer environments, resources, tools, documentation and content for you to take full advantage of the power of Force.com. Begin by following these two steps:

- Get a **Free Developer Edition**
- Browse **Getting Started**

Special
for Deve

The Cloud Compu
August 30-Septembe

Click to Learn M

Technical Library

Access reference documentation, core articles and tools.

- **Documentation**
- **Core Resources**
- **Tools**

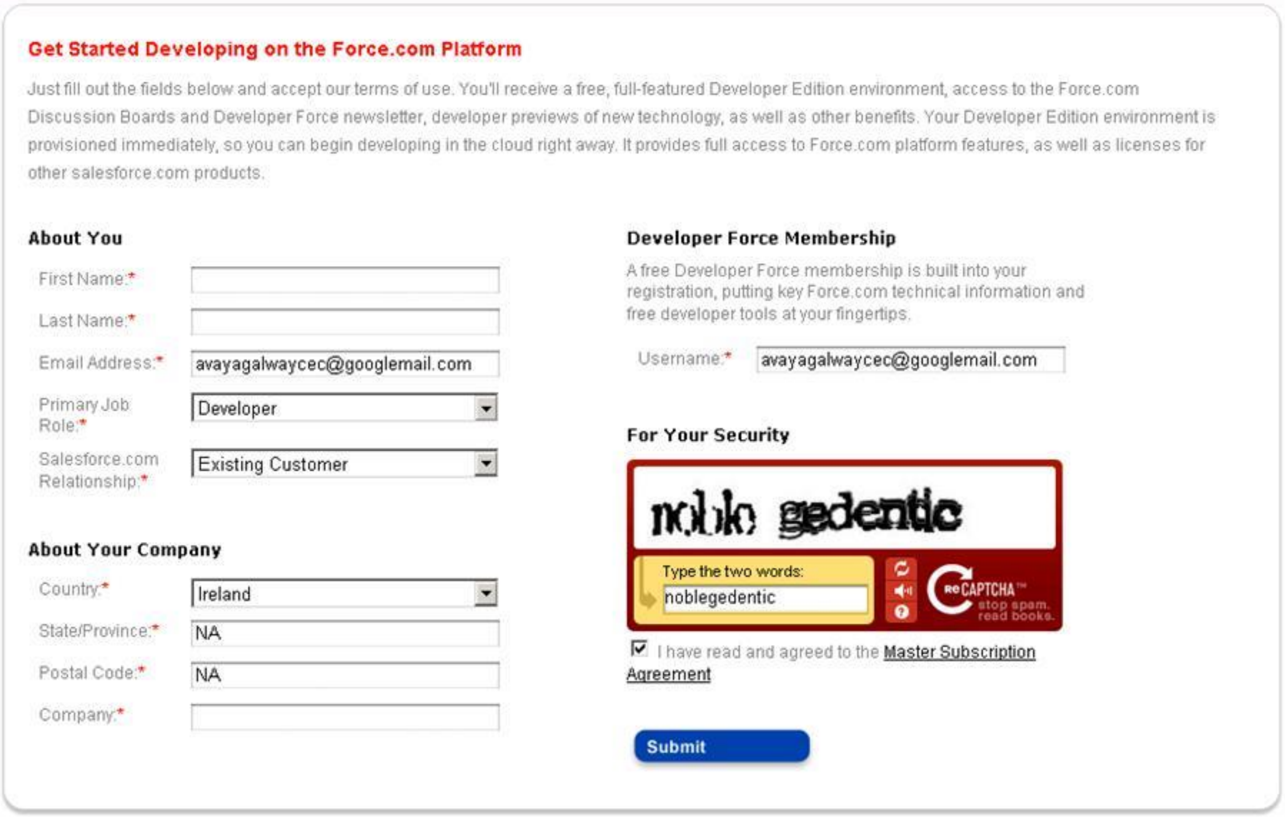
Browse resources by functional

Latest Featured

Single Sign-On with For and Microsoft Active D Federation Services

How to configure seam login from the Windows to Force.com using SS SAML.

You will now see the following screen where you have to enter your company details plus a valid email address which you will be sent your Salesforce username and password.



Get Started Developing on the Force.com Platform

Just fill out the fields below and accept our terms of use. You'll receive a free, full-featured Developer Edition environment, access to the Force.com Discussion Boards and Developer Force newsletter, developer previews of new technology, as well as other benefits. Your Developer Edition environment is provisioned immediately, so you can begin developing in the cloud right away. It provides full access to Force.com platform features, as well as licenses for other salesforce.com products.

About You

First Name:*	
Last Name:*	
Email Address:*	avayagalwaycec@googlemail.com
Primary Job Role:*	Developer
Salesforce.com Relationship:*	Existing Customer

About Your Company

Country:*	Ireland
State/Province:*	NA
Postal Code:*	NA
Company:*	

Developer Force Membership

A free Developer Force membership is built into your registration, putting key Force.com technical information and free developer tools at your fingertips.

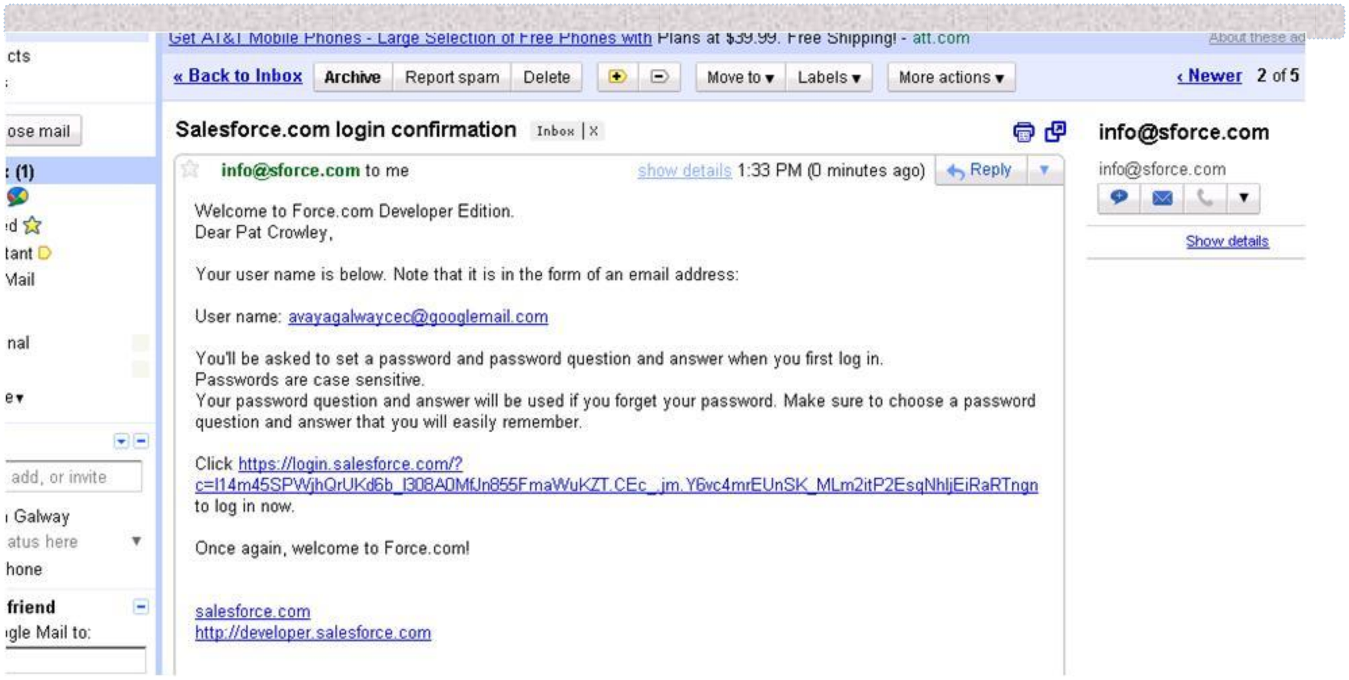
Username:* avayagalwaycec@googlemail.com

For Your Security

Type the two words: noblegedentic

[✓] I have read and agreed to the Master Subscription Agreement

Now login to the email account you provided as part of the registration process to activate your Salesforce developer account. Sample email shown here.



Salesforce.com login confirmation

info@sforce.com to me — 1:33 PM (0 minutes ago)

Welcome to Force.com Developer Edition.
Dear Pat Crowley,

Your user name is below. Note that it is in the form of an email address:

User name: avayagalwaycec@googlemail.com

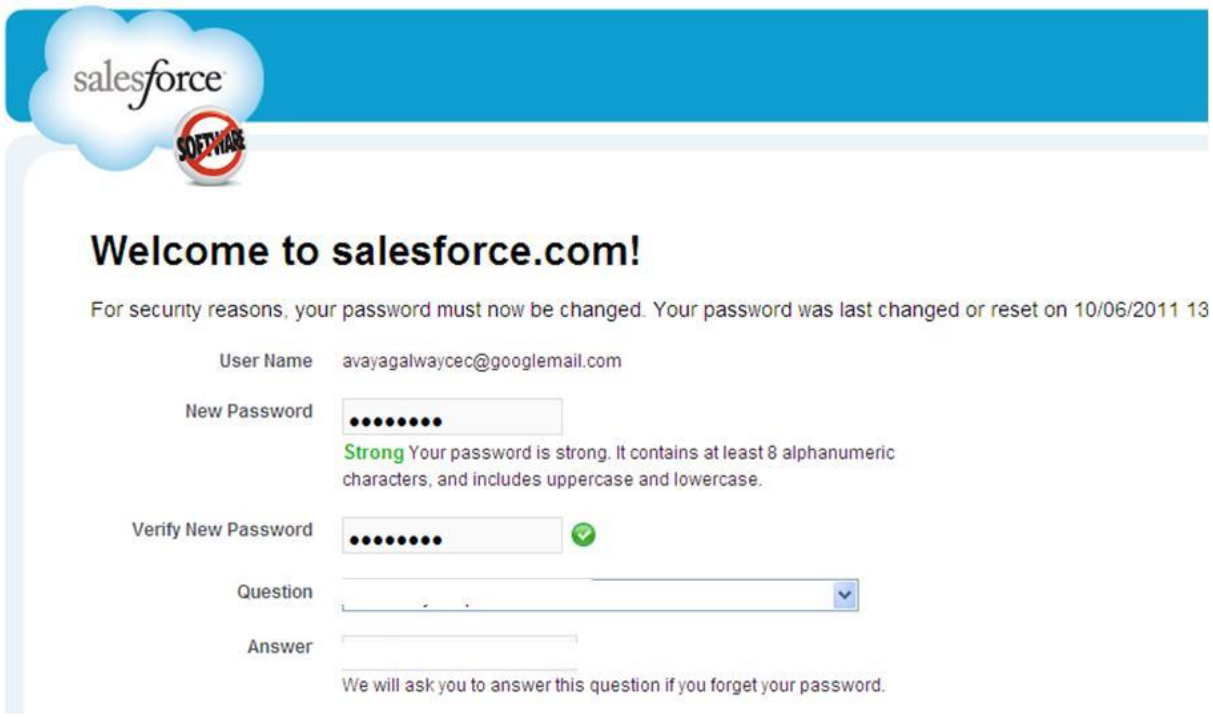
You'll be asked to set a password and password question and answer when you first log in.
Passwords are case sensitive.
Your password question and answer will be used if you forget your password. Make sure to choose a password question and answer that you will easily remember.

Click https://login.salesforce.com/?c=I14m45SPWjhQrUKd6b_I308A0MfJn855FmaWuKZT.CEc_jm.Y6vc4mrEUnSK_MLm2itP2EsqNhljEiRaRTngn to log in now.

Once again, welcome to Force.com!

salesforce.com
http://developer.salesforce.com

In the email, you will click on the embedded link to activate your Salesforce account and set your password. Once completed then login to your Salesforce developer account.



Welcome to salesforce.com!

For security reasons, your password must now be changed. Your password was last changed or reset on 10/06/2011 13

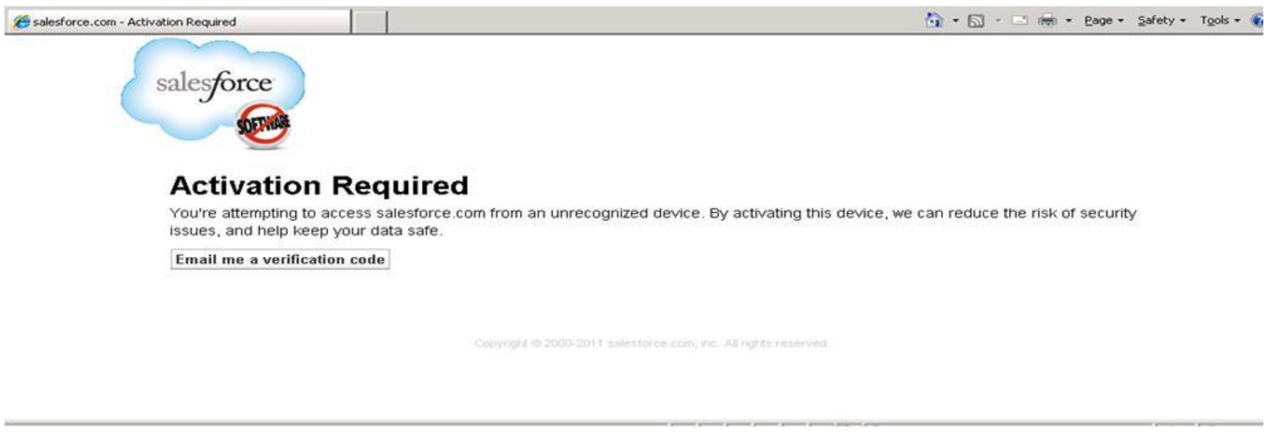
User Name	avayagalwaycec@googlemail.com
New Password	••••••••
	Strong Your password is strong. It contains at least 8 alphanumeric characters, and includes uppercase and lowercase.
Verify New Password	••••••••
Question	
Answer	
	We will ask you to answer this question if you forget your password.

Once you configure your new password, you should then be brought into your Salesforce account where you can begin to create applications and accounts.

Enterprise Server/Workstation Salesforce Authorisation Procedure

Depending on your organisation's integration to Salesforce, you may or may not have to perform the next step on the server/workstation that you are accessing Salesforce. For large enterprise deployments, there are procedures available from Salesforce which allow IP address ranges to bypass the following optional step. If you wish to bypass this step, then instructions are provided in the appendix at the end of this application note.

You may now see the following screen (this depends on whether Salesforce were ever accessed from this particular client PC/workstation before)



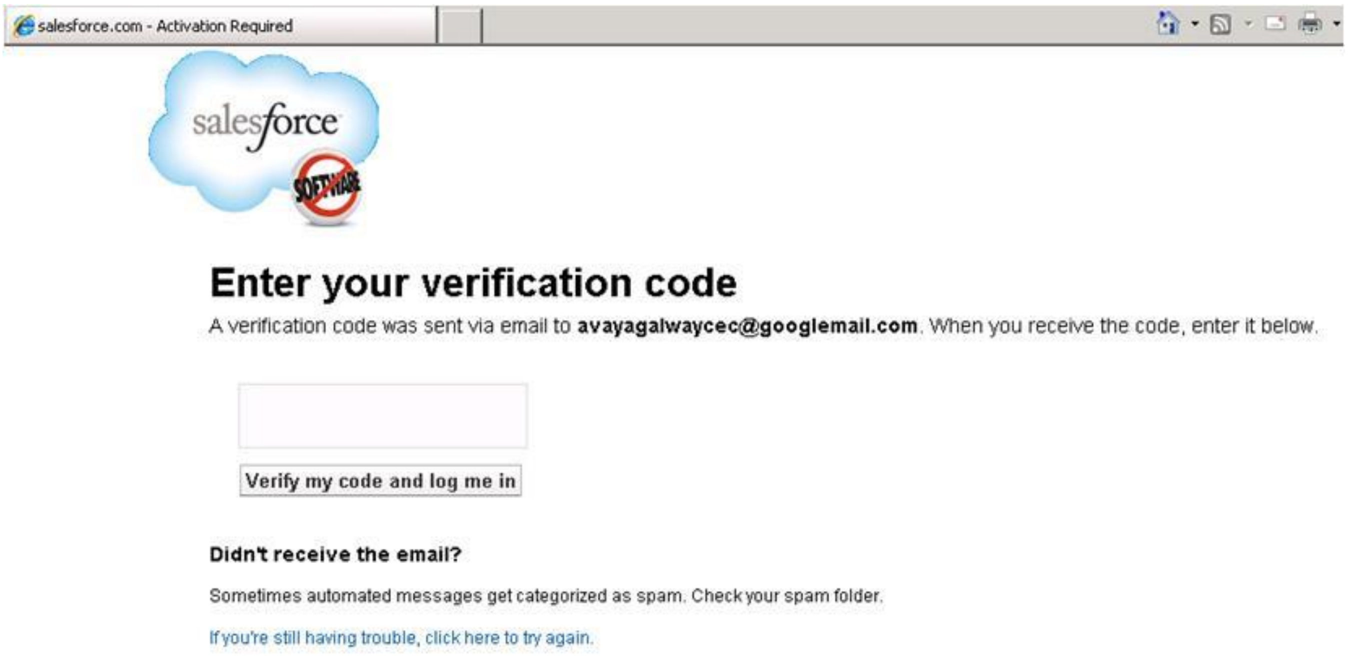
Activation Required

You're attempting to access salesforce.com from an unrecognized device. By activating this device, we can reduce the risk of security issues, and help keep your data safe.

Email me a verification code

Copyright © 2000-2011 salesforce.com, inc. All rights reserved.

If you do see the above screen, then follow the instructions to activate your client PC/workstation. Click on "Email me a verification code" and you will be presented with the following screen.



salesforce.com - Activation Required

Enter your verification code

A verification code was sent via email to **avayagalwaycec@googlemail.com**. When you receive the code, enter it below.

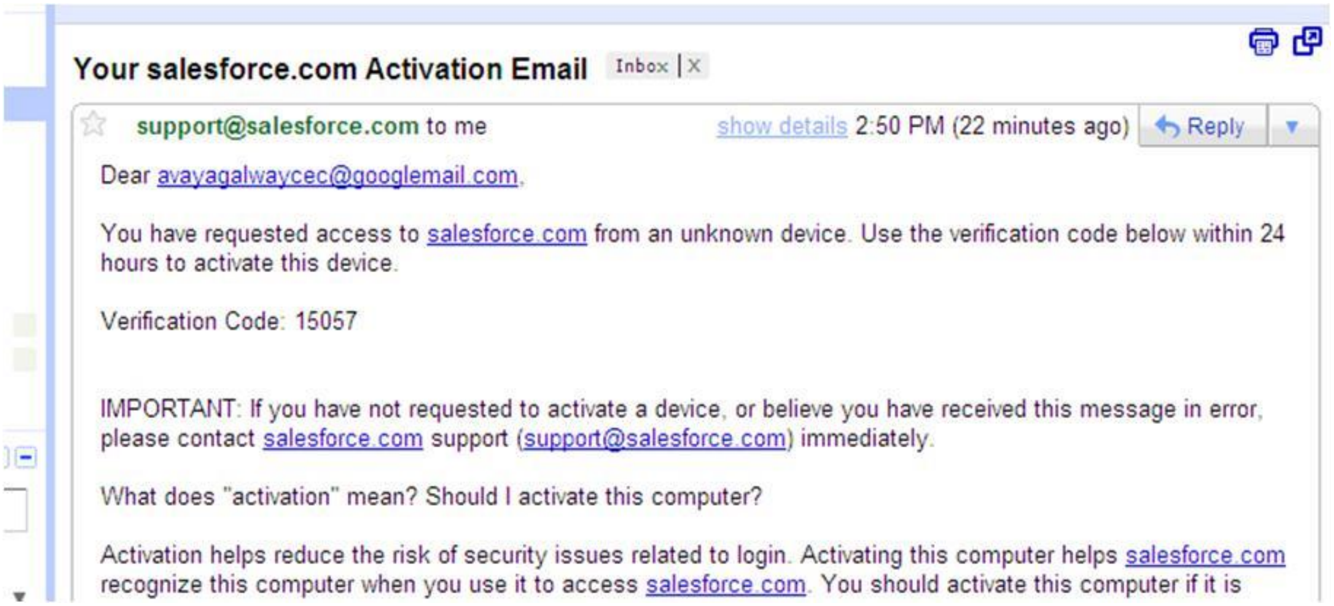
Verify my code and log me in

Didn't receive the email?

Sometimes automated messages get categorized as spam. Check your spam folder.

If you're still having trouble, click here to try again.

To retrieve your activation code, logon to your email account associated with your Salesforce account.



Your salesforce.com Activation Email Inbox | X

support@salesforce.com to me — show details 2:50 PM (22 minutes ago) Reply

Dear avayagalwaycec@googlemail.com,

You have requested access to salesforce.com from an unknown device. Use the verification code below within 24 hours to activate this device.

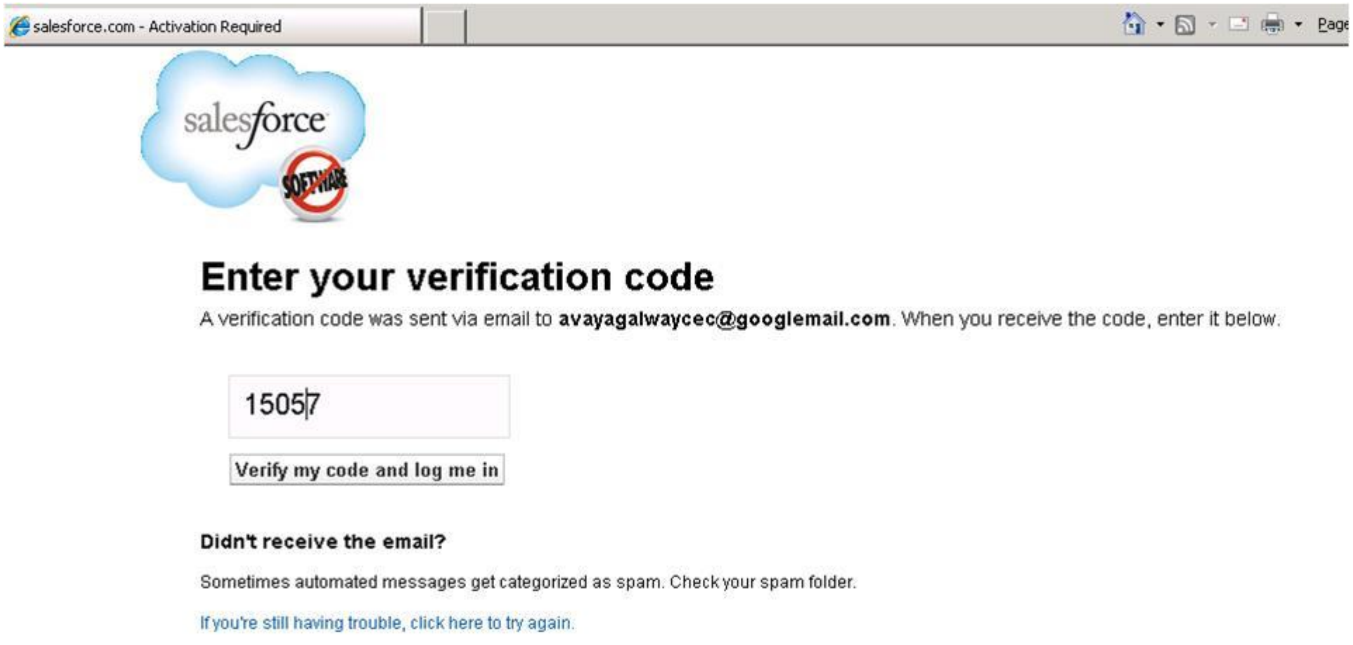
Verification Code: 15057

IMPORTANT: If you have not requested to activate a device, or believe you have received this message in error, please contact salesforce.com support (support@salesforce.com) immediately.

What does "activation" mean? Should I activate this computer?

Activation helps reduce the risk of security issues related to login. Activating this computer helps salesforce.com recognize this computer when you use it to access salesforce.com. You should activate this computer if it is

Enter into the Salesforce screen on your workstation/PC client.



salesforce.com - Activation Required

Enter your verification code

A verification code was sent via email to **avayagalwaycec@googlemail.com**. When you receive the code, enter it below.

15057

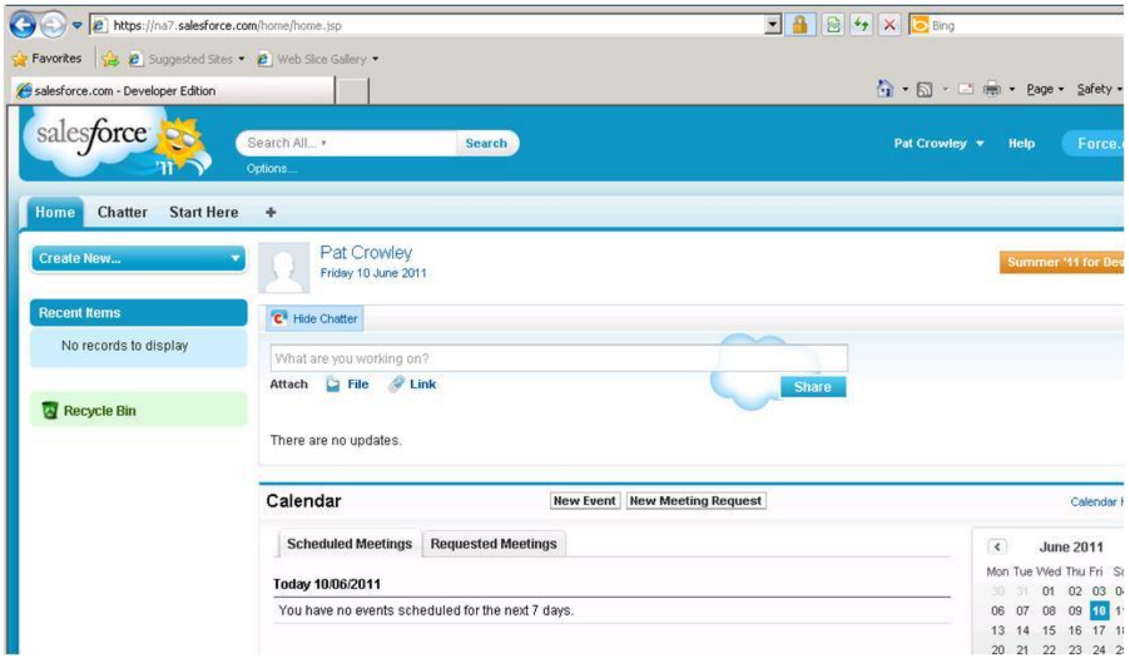
Verify my code and log me in

Didn't receive the email?

Sometimes automated messages get categorized as spam. Check your spam folder.

If you're still having trouble, click here to try again.

If you have entered everything correctly then you should arrive here.



https://na7.salesforce.com/home/home.jsp

salesforce.com - Developer Edition

Pat Crowley
Friday 10 June 2011

Recent Items
No records to display

What are you working on?

There are no updates.

Calendar

Scheduled Meetings | Requested Meetings

Today 10/06/2011
You have no events scheduled for the next 7 days.

June 2011

At this stage you are now ready to begin the installation of the Avaya created APEX plug-ins that enables integration from Avaya Contact Center and Salesforce. These integrations have been developed by Avaya for enabling information to be screen popped into the Avaya Contact Center agent desktop Agent Desktop.

The following steps guide you through the installation of the Avaya Apex plug-in. The plug-in itself was updated with some new enhancements as of December 2011 (***Updated December 2011 to cater for 10 digit telephone numbers). Existing customers who have already installed the plug-in will now have to uninstall the plug-in from the Salesforce Developer environment and re-install it using the link below.***

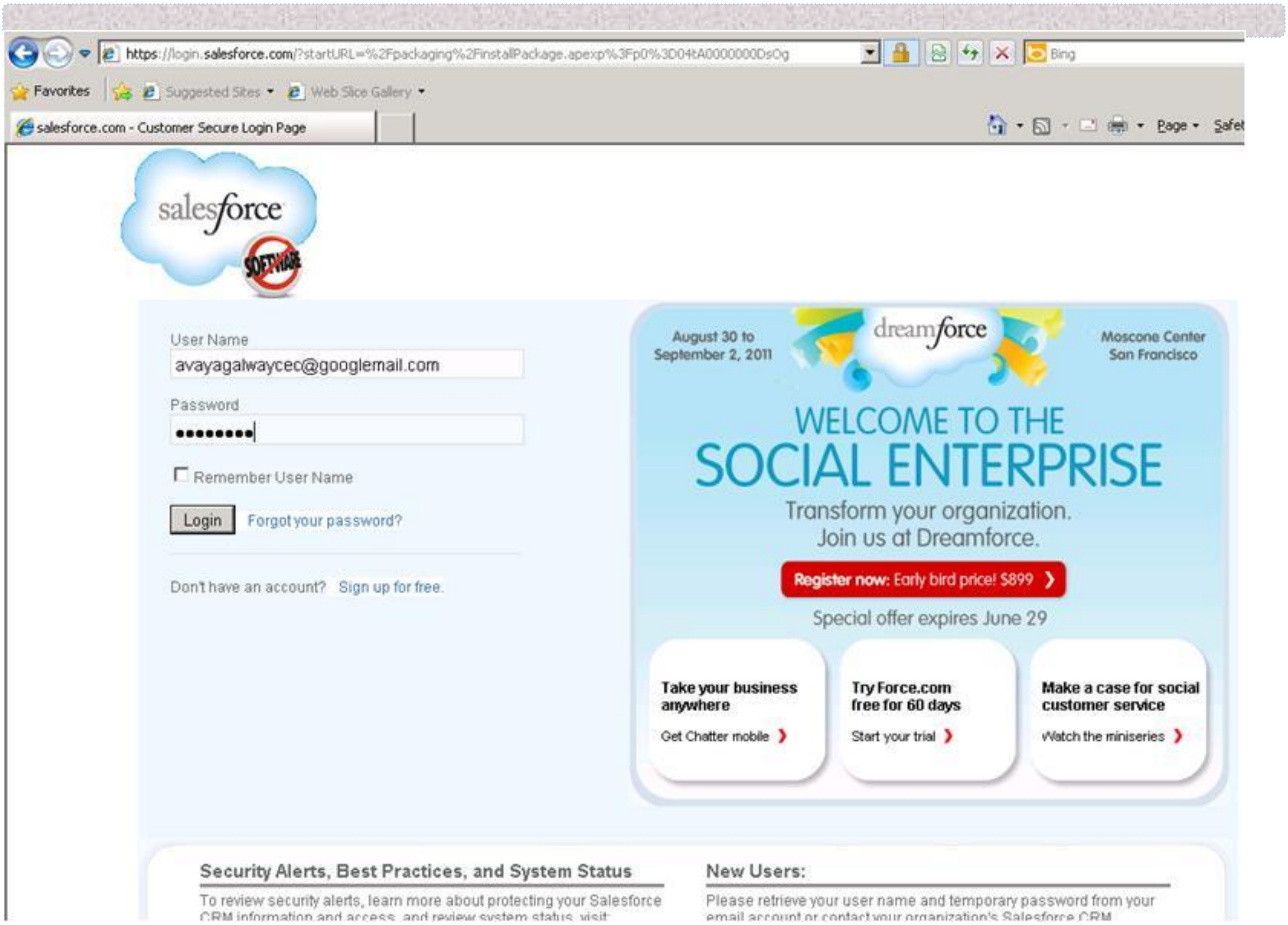
To uninstall an existing deployed application login to your Salesforce Developer account, Select "Setup" from the dropdown menu under your login name and then click on "Installed Packages". Select the existing Avaya plug-in and then select "uninstall". Follow all steps to ensure that the existing package is uninstalled successfully.

Note also that if you have created a custom email field within your Salesforce development environment, you will have to rename the existing custom email field and then the install should proceed correctly.

While logged into your Salesforce developer account (you will only have to do this once if first time to install), then using the following Avaya provided web link, browse to the location as follows.

https://login.salesforce.com/packaging/installPackage.apexp?p0=04tA00000007Uau

You will now be taken to the following installation screen and you will have to login again using your developer account creditentials (even if you were already logged in!). Enter your developer account details and click on login button. It is important to login immediately and complete this step in the same window that has just been opened in your browser.


https://login.salesforce.com/?startURL=%2Fpackaging%2FinstallPackage.apexp%3Fp0%3D04tA0000000DsOg
Favorites
Suggested Sites
Web Slice Gallery
salesforce.com - Customer Secure Login Page
Page
Safet
salesforce

User Name
avayagalwaycec@googlemail.com

Password
••••••••

☐ Remember User Name

Login Forgot your password?

Don't have an account? Sign up for free.

August 30 to September 2, 2011
dreamforce
Moscone Center San Francisco

WELCOME TO THE SOCIAL ENTERPRISE

Transform your organization. Join us at Dreamforce.

Register now: Early bird price! $899

Special offer expires June 29

Take your business anywhere
Get Chatter mobile

Try Force.com free for 60 days
Start your trial

Make a case for social customer service
Watch the miniseries

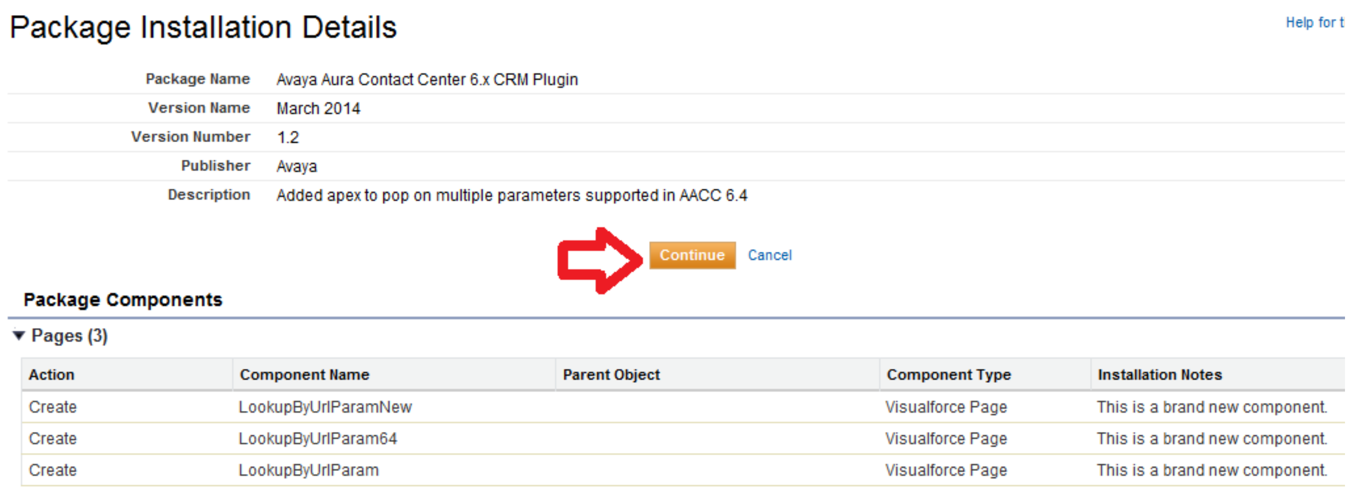
Security Alerts, Best Practices, and System Status

To review security alerts, learn more about protecting your Salesforce CRM information and access, and review system status, visit

New Users:

Please retrieve your user name and temporary password from your email account or contact your organization's Salesforce CRM

Once logged in, you will see the following important screen. This is the famous original Avaya Contact Center 6.x Salesforce plug-in. This must be installed for all Avaya Contact Center Salesforce integrations otherwise it's not out of the box screen pops. Click on Continue to install the Avaya Avaya Contact Center plug-in to your newly created developer account which contains the customer details.



Package Installation Details

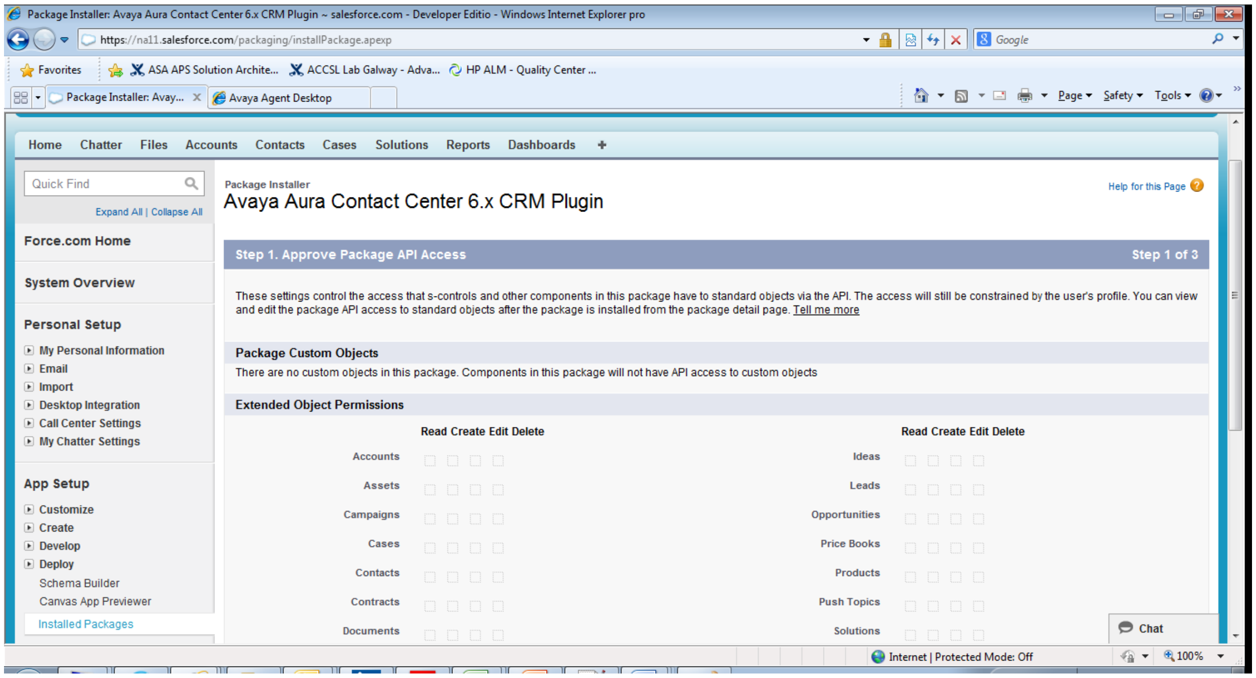
Package Name	Avaya Aura Contact Center 6.x CRM Plugin
Version Name	March 2014
Version Number	1.2
Publisher	Avaya
Description	Added apex to pop on multiple parameters supported in AACC 6.4

Package Components

Pages (3)

Action	Component Name	Parent Object	Component Type	Installation Notes
Create	LookupByUrlParamNew		Visualforce Page	This is a brand new component.
Create	LookupByUrlParam64		Visualforce Page	This is a brand new component.
Create	LookupByUrlParam		Visualforce Page	This is a brand new component.

Click on Continue to arrive at the following screen Step 1 of the 3 part installation process.



Package Installer

Avaya Aura Contact Center 6.x CRM Plugin

Step 1. Approve Package API Access — **Step 1 of 3**

These settings control the access that s-controls and other components in this package have to standard objects via the API. The access will still be constrained by the user's profile. You can view and edit the package API access to standard objects after the package is installed from the package detail page. Tell me more

Package Custom Objects

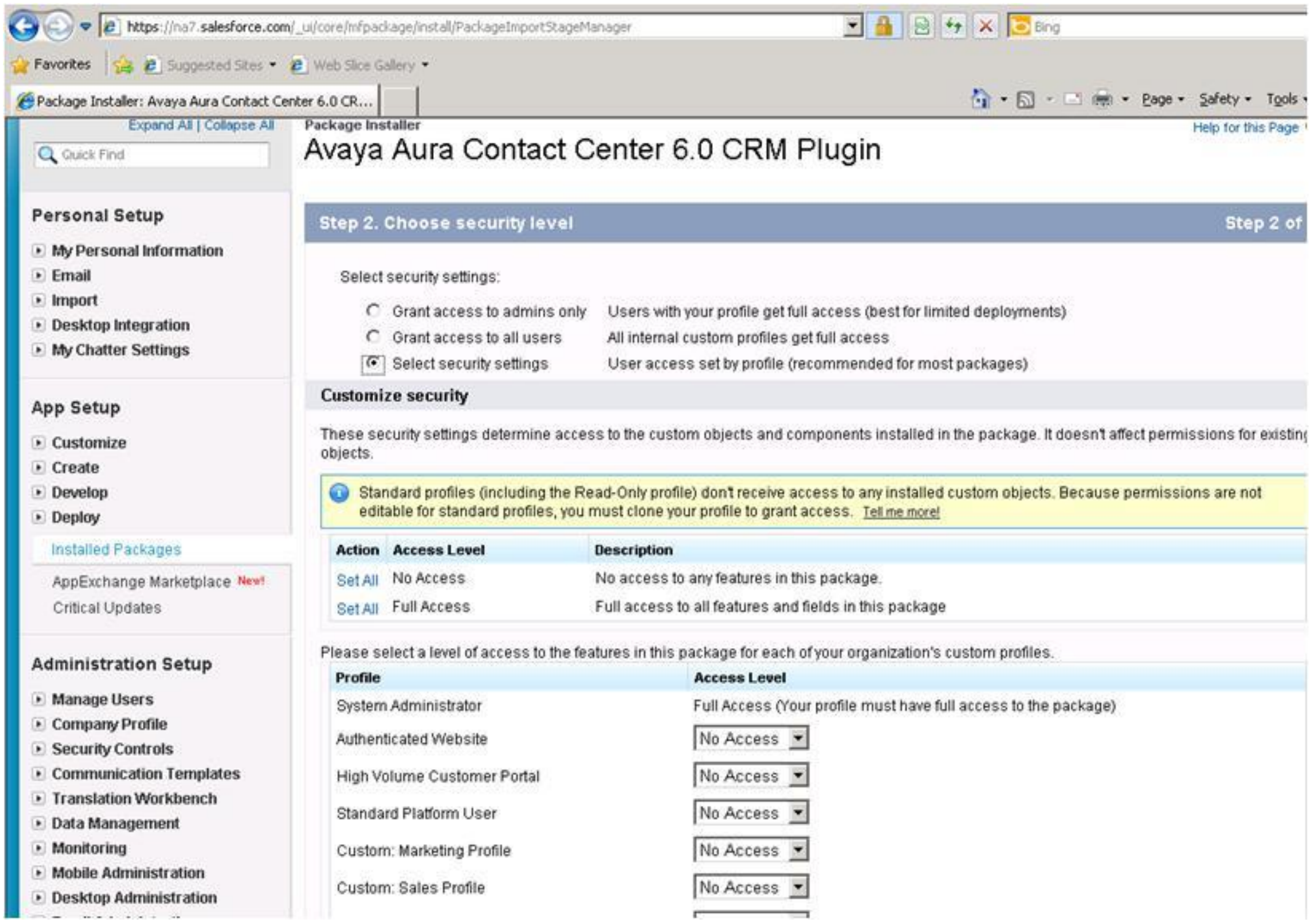
There are no custom objects in this package. Components in this package will not have API access to custom objects

Extended Object Permissions

	Read	Create	Edit	Delete		Read	Create	Edit	Delete
Accounts	☐	☐	☐	☐	Ideas	☐	☐	☐	☐
Assets	☐	☐	☐	☐	Leads	☐	☐	☐	☐
Campaigns	☐	☐	☐	☐	Opportunities	☐	☐	☐	☐
Cases	☐	☐	☐	☐	Price Books	☐	☐	☐	☐
Contacts	☐	☐	☐	☐	Products	☐	☐	☐	☐
Contracts	☐	☐	☐	☐	Push Topics	☐	☐	☐	☐
Documents	☐	☐	☐	☐	Solutions	☐	☐	☐	☐

Scroll down on the right hand side and then click “Next” to move to step 2 of the installation process.

In the screen below for step 2, configure the security settings according to your enterprise guidelines. For a lab install, select the option shown below.


https://na7.salesforce.com/_ui/core/mfpackage/install/PackageImportStageManager

Package Installer: Avaya Aura Contact Center 6.0 CR...

Package Installer

Avaya Aura Contact Center 6.0 CRM Plugin

Step 2. Choose security level — Step 2 of

Select security settings:

( ) Grant access to admins only	Users with your profile get full access (best for limited deployments)
( ) Grant access to all users	All internal custom profiles get full access
[✓] Select security settings	User access set by profile (recommended for most packages)

Customize security

These security settings determine access to the custom objects and components installed in the package. It doesn't affect permissions for existing objects.

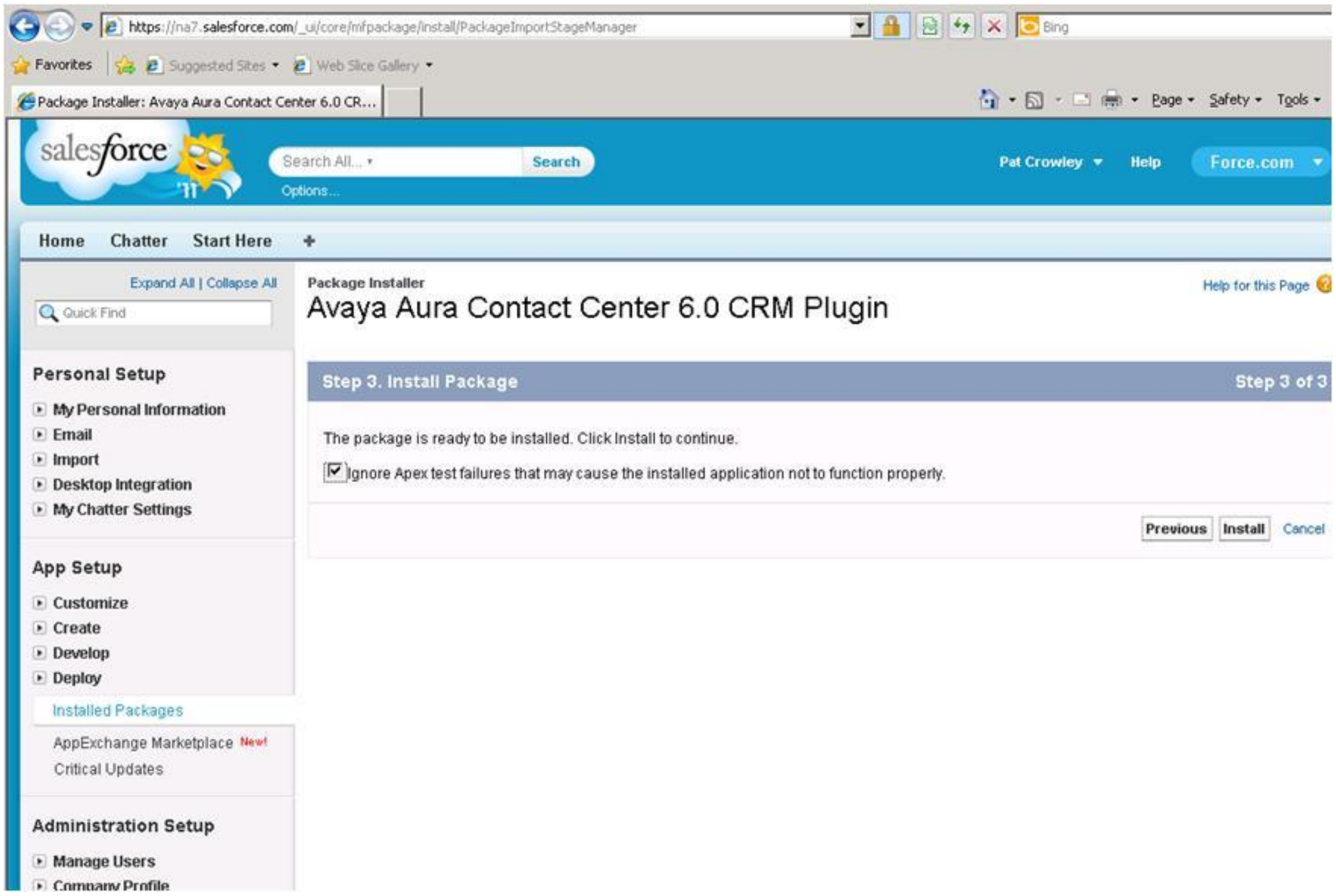
Standard profiles (including the Read-Only profile) don't receive access to any installed custom objects. Because permissions are not editable for standard profiles, you must clone your profile to grant access. Tell me more!

Action	Access Level	Description
Set All	No Access	No access to any features in this package.
Set All	Full Access	Full access to all features and fields in this package

Please select a level of access to the features in this package for each of your organization's custom profiles.

Profile	Access Level
System Administrator	Full Access (Your profile must have full access to the package)
Authenticated Website	No Access
High Volume Customer Portal	No Access
Standard Platform User	No Access
Custom: Marketing Profile	No Access
Custom: Sales Profile	No Access

Now click Next to move to Step 3 of 3.



Package Installer

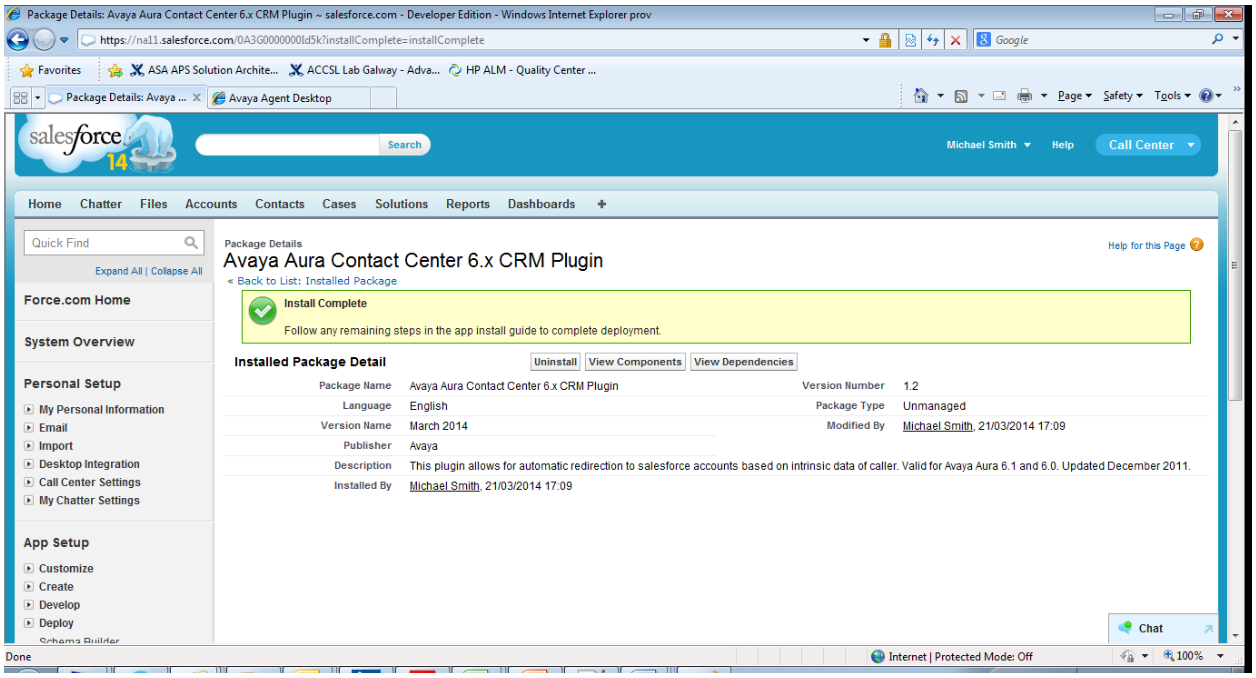
Avaya Aura Contact Center 6.0 CRM Plugin

Step 3. Install Package — Step 3 of 3

The package is ready to be installed. Click Install to continue.

[✓] Ignore Apex test failures that may cause the installed application not to function properly.

Tick the option shown above and then click on Install to complete the installation.


Package Details: Avaya Aura Contact Center 6.x CRM Plugin ~ salesforce.com - Developer Edition - Windows Internet Explorer prov
https://na11.salesforce.com/0A3G0000000Id5k?installComplete=installComplete
Favorites
ASA APS Solution Archite...
ACCSL Lab Galway - Adva...
HP ALM - Quality Center ...
Package Details: Avaya ...
Avaya Agent Desktop
Page
Safety
Tools
Search
Michael Smith
Help
Call Center
Home
Chatter
Files
Accounts
Contacts
Cases
Solutions
Reports
Dashboards
Quick Find
Expand All | Collapse All
Force.com Home
System Overview
Personal Setup
My Personal Information
Email
Import
Desktop Integration
Call Center Settings
My Chatter Settings
App Setup
Customize
Create
Develop
Deploy
Schema Builder
Chat
Done
Internet | Protected Mode: Off
100%

Package Details

Avaya Aura Contact Center 6.x CRM Plugin

« Back to List: Installed Package

Help for this Page

Install Complete

Follow any remaining steps in the app install guide to complete deployment.

Installed Package Detail Uninstall | View Components | View Dependencies

Package Name	Avaya Aura Contact Center 6.x CRM Plugin	Version Number	1.2
Language	English	Package Type	Unmanaged
Version Name	March 2014	Modified By	Michael Smith, 21/03/2014 17:09
Publisher	Avaya		
Description	This plugin allows for automatic redirection to salesforce accounts based on intrinsic data of caller. Valid for Avaya Aura 6.1 and 6.0. Updated December 2011.		
Installed By	Michael Smith, 21/03/2014 17:09		

If you see the above screen, then you have successfully installed the Avaya Contact Center 6.x Salesforce "plug-in" which contains the Salesforce Apex code modules which we will use later on in this document. If you wish to see the Apex modules that are installed, then click on "View Components".

Name:	**Description**
LookupByUrlParmController	Apex class takes as data input from Avaya Contact Center (calling number or Account number) and returns a Salesforce account name which is then used to deliver a screen pop of the relevant account or contact onto the agent workstation.
LookupByUrlParmControllerNew	Extends the functionality delivered in LookupByUrlParmController to include searches on email and website.
LookupByUrlParmController64	Extends LookupByUrlParmController further to accept parameters (calling number, customer number, phone or email) individually or collectively. Agent Desktop supports passing multiple parameters since the release of Avaya Contact Center 6.4.
LookupByUrlParm	Visualforce page used to call the apex code LookupByUrlParmController
LookupByUrlParmNew	Visualforce page used to call the apex code LookupByUrlParmControllerNew.
LookupByUrlParm64	Visualforce page used to call the apex code LookupByUrlParmController64.
LookupByUrl64TestSuite	Apex class containing test methods for LookupByUrlParmController64

Appendix 2 at the end of this document gives the complete set of code instructions for the Apex classes listed above.

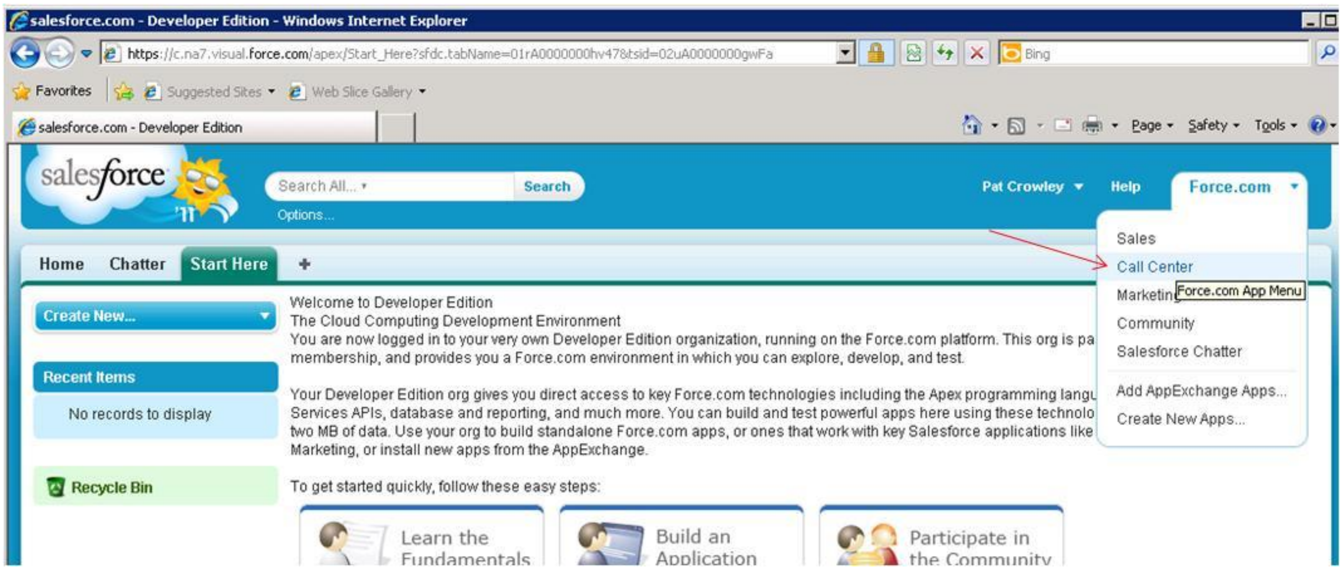
Note: When a Salesforce.com APP is developed, a package is created which has a link for each app you develop in your account. Avaya has created the above link which allows the Sample Avaya app to be available to all other developer accounts. Avaya has not published their sample app to the Salesforce App store.

The visual force page links will be unique for all customers. You can login to your Salesforce developer account to see how the URL will be formatted. The "na7" or "na11" in the screenshots are for our (Avaya) instances of accessing Salesforce.

The apex code can be saved and opened in any text editor and reused. Later in this white paper we show how to create new Apex classes for advanced integration.
Avaya Contact Center

Creating Salesforce Customer Records

To test the integration, you will have to create a number of test customer accounts within your Salesforce account. To do this select the "Force.com" option in top right hand side and select Call Center as shown here.

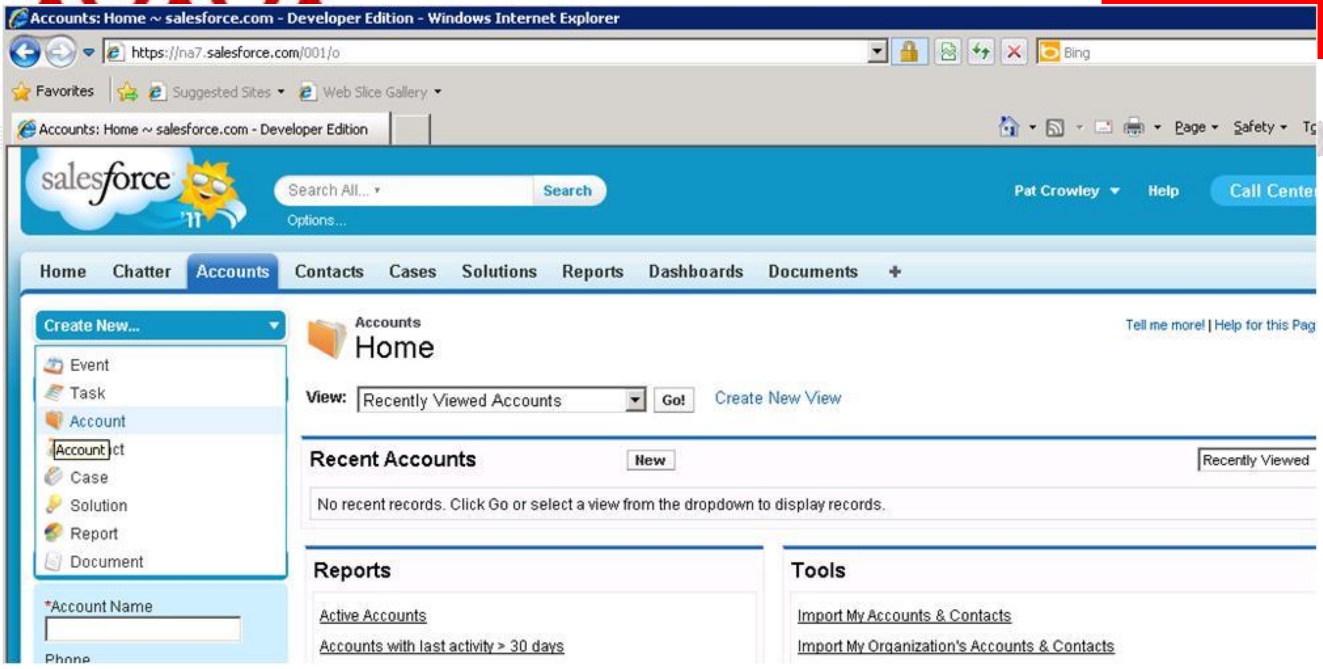


Welcome to Developer Edition
The Cloud Computing Development Environment
You are now logged in to your very own Developer Edition organization, running on the Force.com platform. This org is pa
membership, and provides you a Force.com environment in which you can explore, develop, and test.

Your Developer Edition org gives you direct access to key Force.com technologies including the Apex programming langu
Services APIs, database and reporting, and much more. You can build and test powerful apps here using these technolo
two MB of data. Use your org to build standalone Force.com apps, or ones that work with key Salesforce applications like
Marketing, or install new apps from the AppExchange.

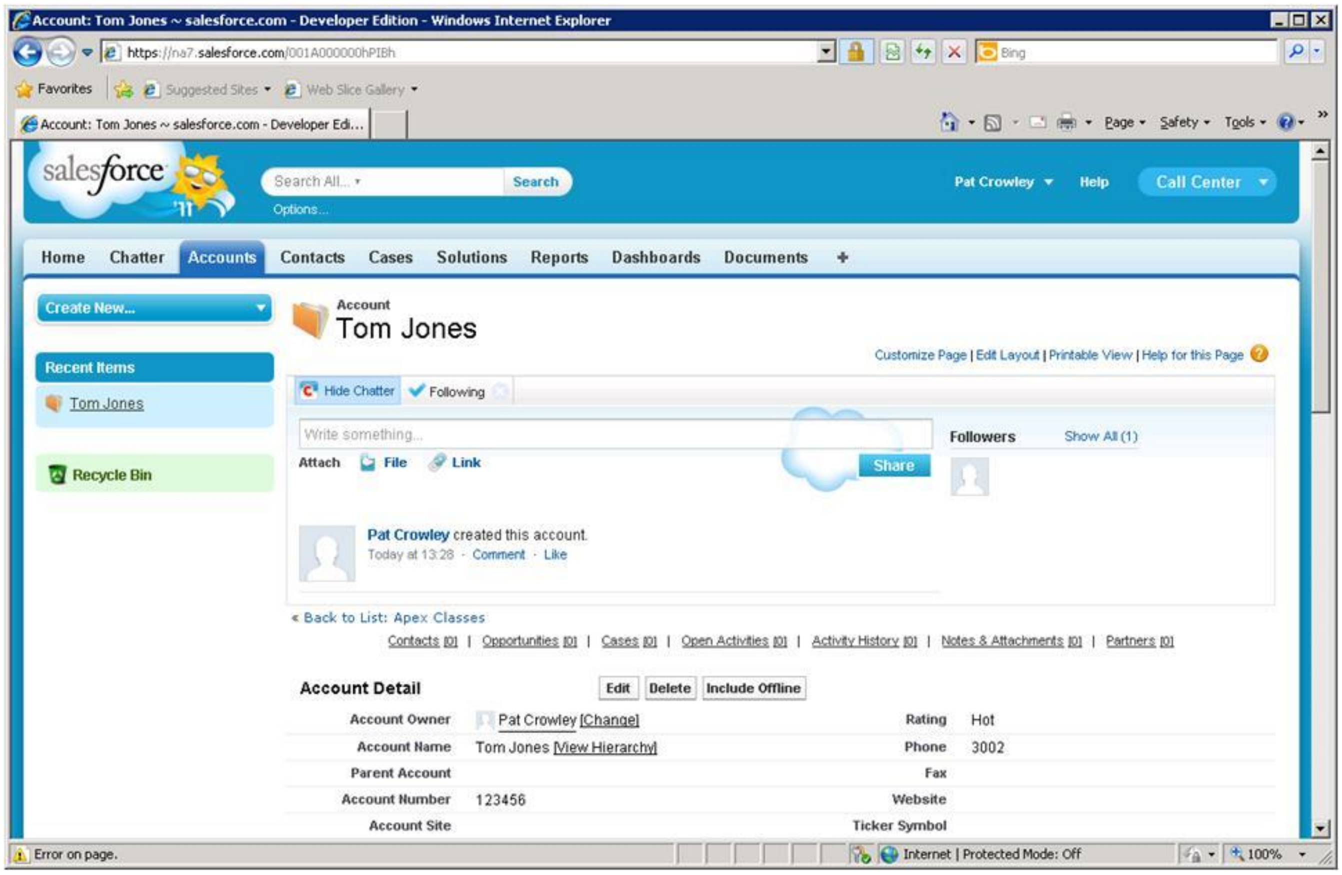
To get started quickly, follow these easy steps:

Select Accounts in tool bar and then select Account.



No recent records. Click Go or select a view from the dropdown to display records.

In our example we have created the account for customer Tom Jones with phone number 3002 as follows.


Account: Tom Jones ~ salesforce.com - Developer Edition - Windows Internet Explorer
https://na7.salesforce.com/001A000000hPIBh
Search All...
Search
Pat Crowley
Help
Call Center
Home
Chatter
Accounts
Contacts
Cases
Solutions
Reports
Dashboards
Documents
Create New...
Recent Items
Tom Jones
Recycle Bin
Account
Tom Jones
Customize Page | Edit Layout | Printable View | Help for this Page
Hide Chatter
Following
Write something...
Attach
File
Link
Share
Followers
Show All (1)
« Back to List: Apex Classes
Contacts [0] | Opportunities [0] | Cases [0] | Open Activities [0] | Activity History [0] | Notes & Attachments [0] | Partners [0]
Edit
Delete
Include Offline
Error on page.
Internet | Protected Mode: Off
100%

Pat Crowley created this account.
Today at 13:28 · Comment · Like

Account Detail

Account Owner	Pat Crowley [Change]	Rating	Hot
Account Name	Tom Jones [View Hierarchy]	Phone	3002
Parent Account		Fax	
Account Number	123456	Website	
Account Site		Ticker Symbol	

In the next chapter we will conclude the client side integration by describing the configuration required on Avaya Contact Center to enable a screen pop on the telephone number using the basic screen popping functionality.

Also detailed is screen popping on both the telephone and account numbers combined using the Agent Desktop advanced screen pop wizard delivered as part of Avaya Contact Center 6.4.

4 – CONFIGURING AGENT DESKTOP SCREEN POP

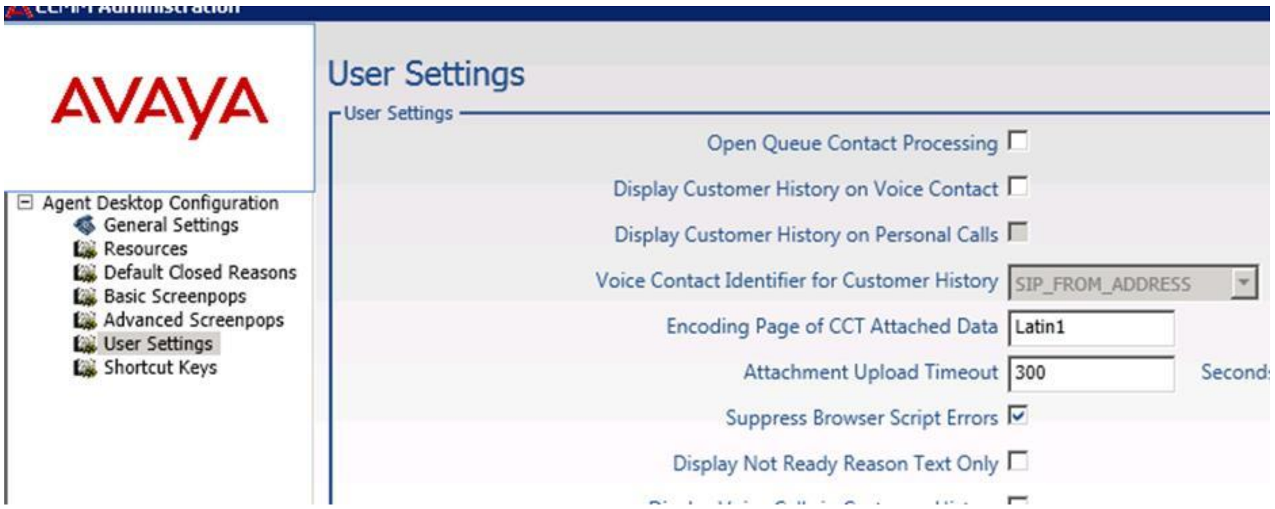
The last step on this client side integration process is to define the actual screen pop of the customer account from Salesforce once the agent receives or answers a contact. To define the Agent Desktop screen in Avaya Contact Center, parameters are configured using the CCMM administration application. In a voice only solution this admin application is not available and therefore screen popping is not supported.

There are two options for creating a screen pop the first has been available since Avaya Aura® Contact Center releases 6.2 and is detailed in 4A. Called the basic screen pop it is possible to pop on one parameter only, without filtering – pops will occur for each call. This is the only available option to customers on Avaya Aura® Contact Center releases 6.2 and 6.3.
Since the release of Avaya Aura® Contact Center 6.4 advanced screen pop is available and detailed in section 4B below. The advanced option allows filtering of when a pop should occur and multiple parameters can be passed to the URL or customer application.

Note: To set up a screen pop in Agent Desktop follow the instructions in sections 4A or 4B.

To open the CCMM administration application, login to CCMA, click on “multimedia” and accept all of the prompts to download and open the CCMM application.

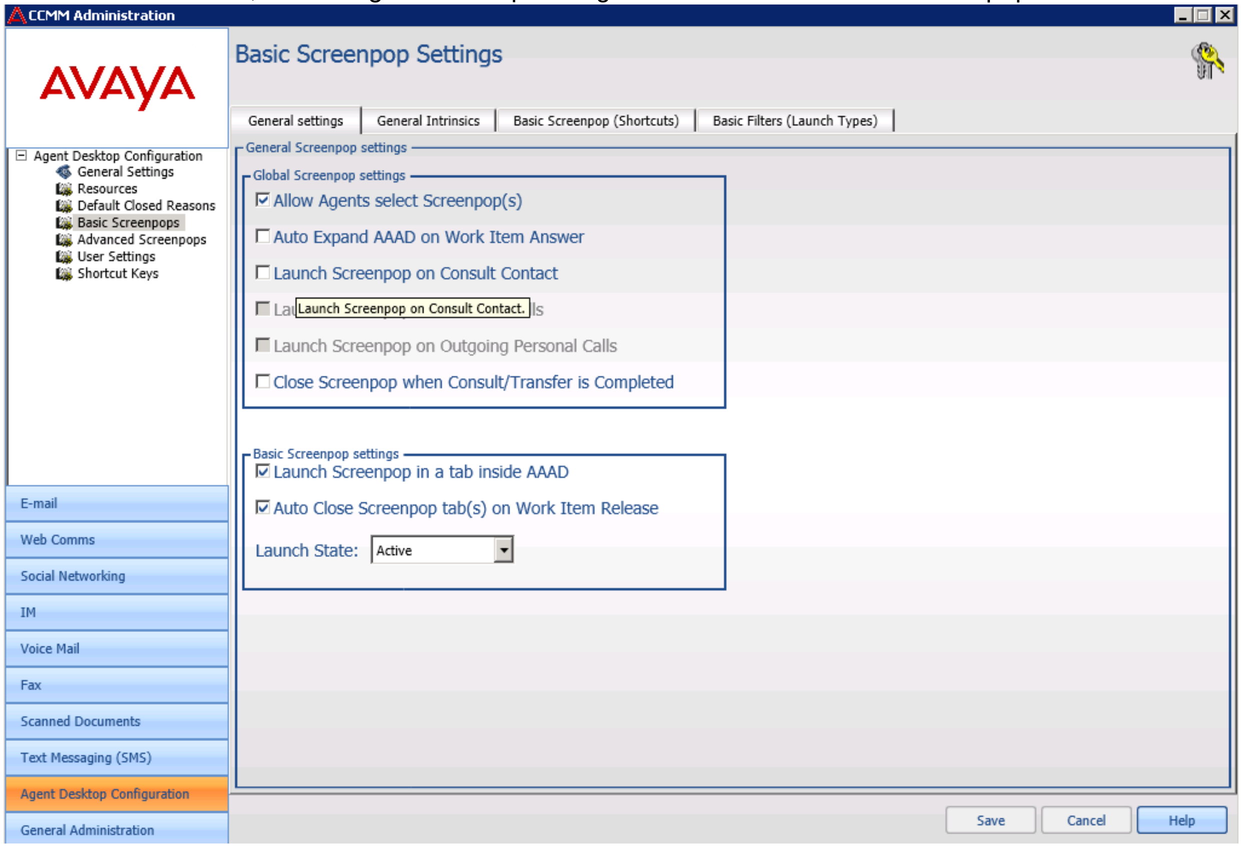
Select “Agent Desktop Configuration” and on “User Settings” form tick “Suppress Browser Script Errors”.



Open Queue Contact Processing	☐
Display Customer History on Voice Contact	☐
Display Customer History on Personal Calls	☐
Voice Contact Identifier for Customer History	SIP_FROM_ADDRESS
Encoding Page of CCT Attached Data	Latin1
Attachment Upload Timeout	300 Second
Suppress Browser Script Errors	☑
Display Not Ready Reason Text Only	☐

4A – SCREENPOP ON CALLING NUMBER ONLY

To screenpop on the calling number only; functionality available prior to Avaya Aura® Contact Center 6.4. On the bottom left hand side, select "Agent Desktop Configuration" and then "Basic Screenpops" as shown here.



Basic Screenpop Settings

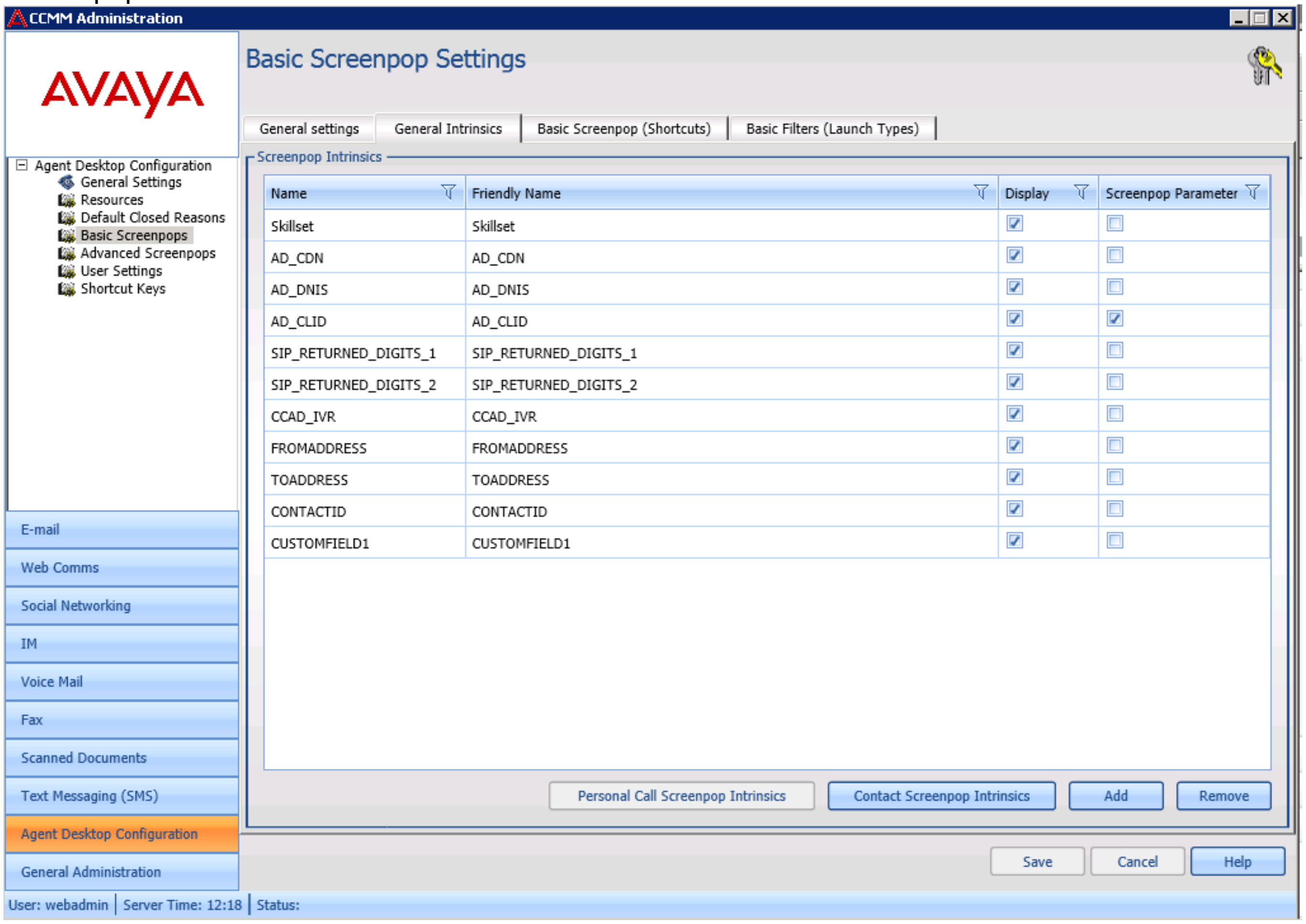
General Screenpop settings

Global Screenpop settings
- [✓] Allow Agents select Screenpop(s)
- [ ] Auto Expand AAAD on Work Item Answer
- [ ] Launch Screenpop on Consult Contact
- [ ] La Launch Screenpop on Consult Contact. ls
- [ ] Launch Screenpop on Outgoing Personal Calls
- [ ] Close Screenpop when Consult/Transfer is Completed

Basic Screenpop settings
- [✓] Launch Screenpop in a tab inside AAAD
- [✓] Auto Close Screenpop tab(s) on Work Item Release

Launch State: Active

On the second tab 'General Intrinsics' select the 'AD_CLID' (calling number) option ticking the Screenpop Parameter column tick box:



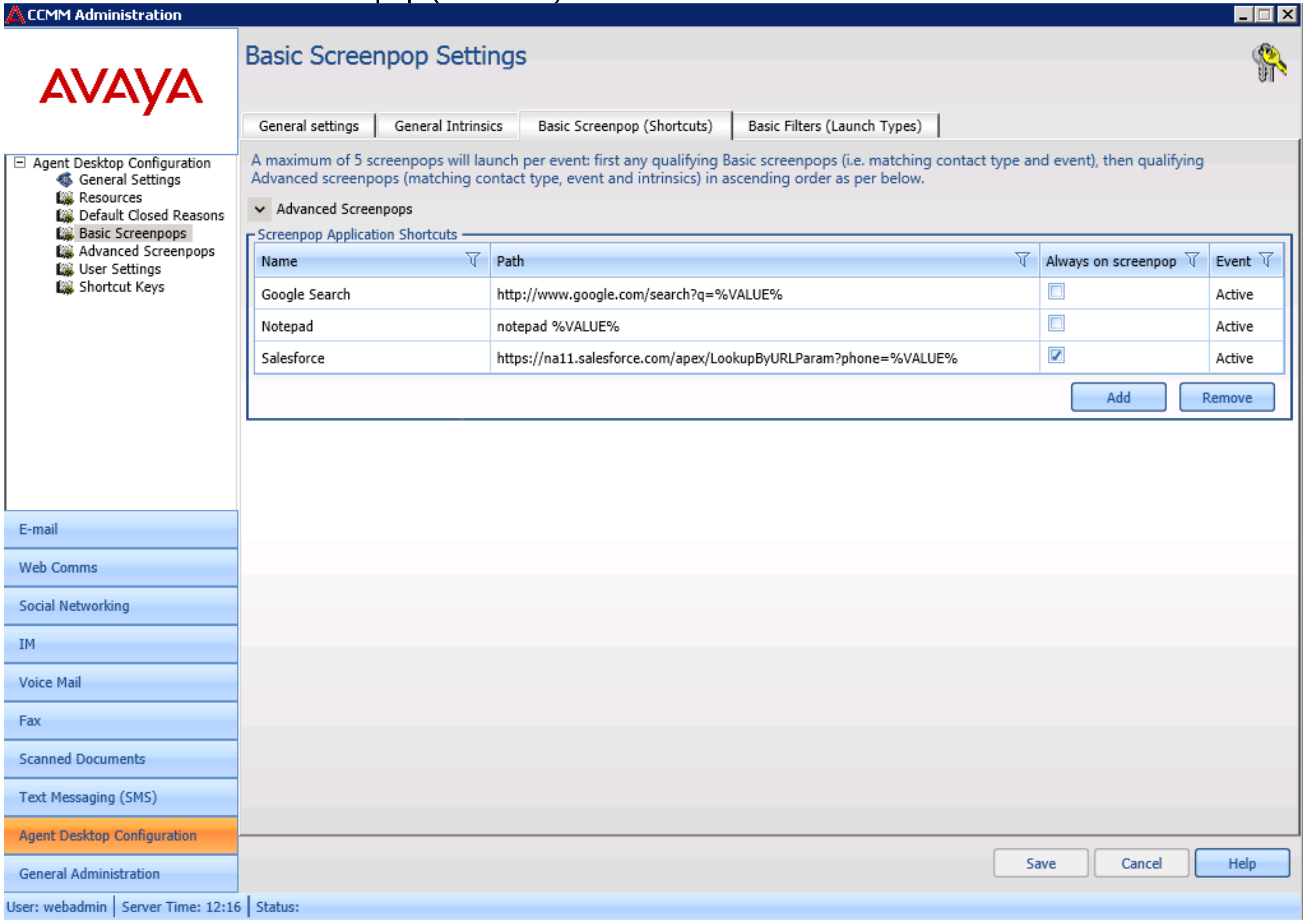
Basic Screenpop Settings

Screenpop Intrinsics

Name	Friendly Name	Display	Screenpop Parameter
Skillset	Skillset	✓	
AD_CDN	AD_CDN	✓	
AD_DNIS	AD_DNIS	✓	
AD_CLID	AD_CLID	✓	✓
SIP_RETURNED_DIGITS_1	SIP_RETURNED_DIGITS_1	✓	
SIP_RETURNED_DIGITS_2	SIP_RETURNED_DIGITS_2	✓	
CCAD_IVR	CCAD_IVR	✓	
FROMADDRESS	FROMADDRESS	✓	
TOADDRESS	TOADDRESS	✓	
CONTACTID	CONTACTID	✓	
CUSTOMFIELD1	CUSTOMFIELD1	✓	

User: webadmin | Server Time: 12:18 | Status:

On the third tab Basic Screenpop (shortcuts) add the Salesforce URL.



Basic Screenpop Settings

A maximum of 5 screenpops will launch per event: first any qualifying Basic screenpops (i.e. matching contact type and event), then qualifying Advanced screenpops (matching contact type, event and intrinsics) in ascending order as per below.

Screenpop Application Shortcuts

Name	Path	Always on screenpop	Event
Google Search	http://www.google.com/search?q=%VALUE%	☐	Active
Notepad	notepad %VALUE%	☐	Active
Salesforce	https://na11.salesforce.com/apex/LookupByURLParam?phone=%VALUE%	✓	Active

User: webadmin | Server Time: 12:16 | Status:

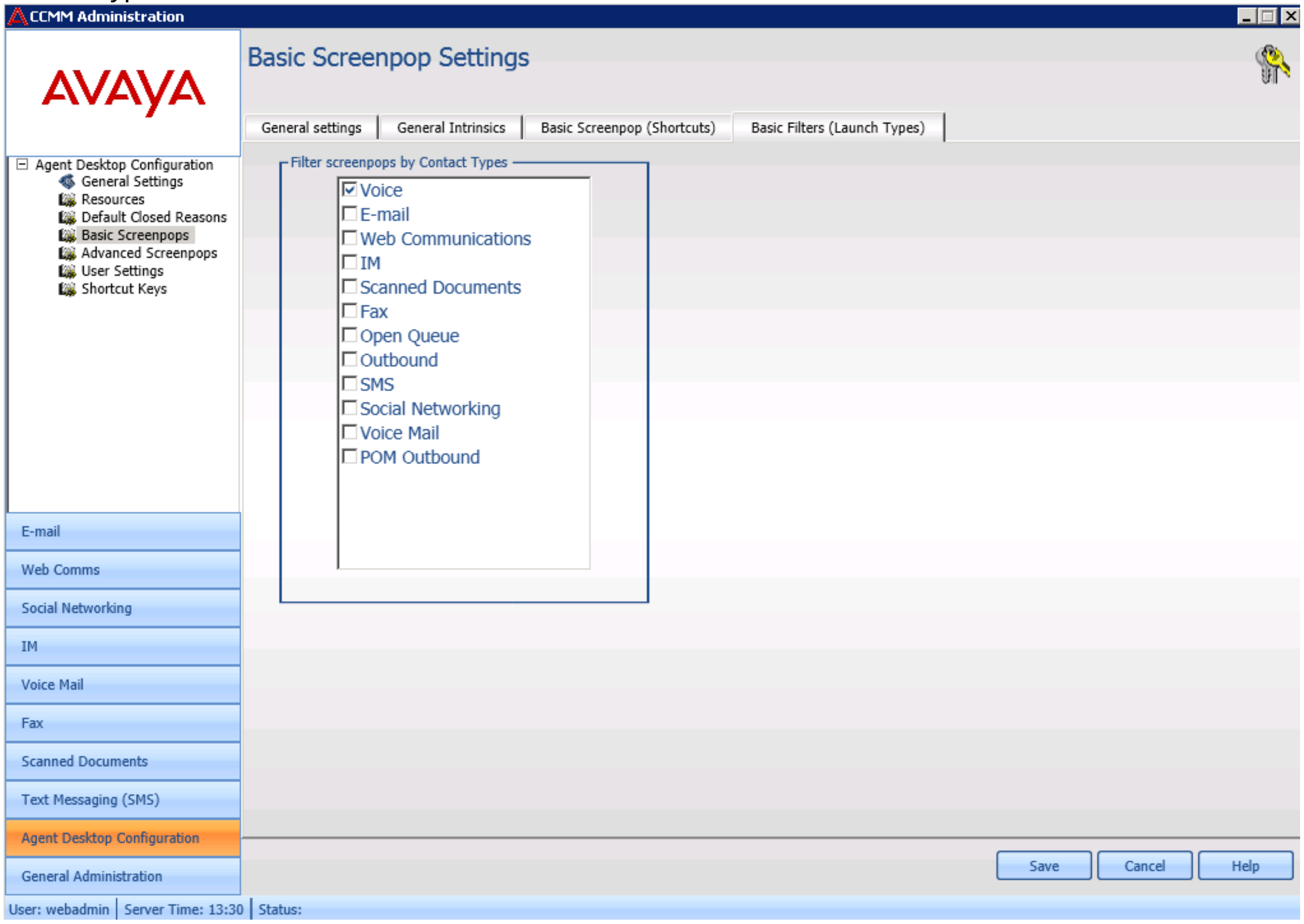
In the above illustration, we have configured 3 different possible screen pop applications. The 3rd one is enabled for our screen pop indicated by the “tick” in the “Always on screenpop” column.

The command https://na7.salesforce.com/apex/LookupByURLParam?phone=%VALUE% is simply a web service call to the Avaya Salesforce APEX class that is defined in the plug-in we previously imported into our Salesforce account. Without this Apex class, we cannot have a screen pop of the customer details. To add this into your Avaya Contact Center installation, click on the “Add” button” in the upper half of the window and simply paste the full URL above into the blank placeholder. Then click Save. For the settings to take effect, the agent must restart Agent Desktop and then login to the contact center.

The %VALUE% parameter is a variable which will contain the customers calling number or A-number if available AD_CLID (calling number) is enabled on the second tab. This means that when the agent receives a voice contact, Agent Desktop will insert the calling party’s phone number into the above Salesforce Apex class. In this way we can search Salesforce for every customer call and screen pop the customer details if a valid match is found in Salesforce. Other parameters such as email address or customer collected digits from an IVR could also be used. So for our installation we have the following command configured in our screen-pop URL:

https://na7.salesforce.com/apex/LookupByURLParam?phone=%VALUE%

The final tab on the Agent desktop configuration allows filtering when screenpops are launched by contact type:



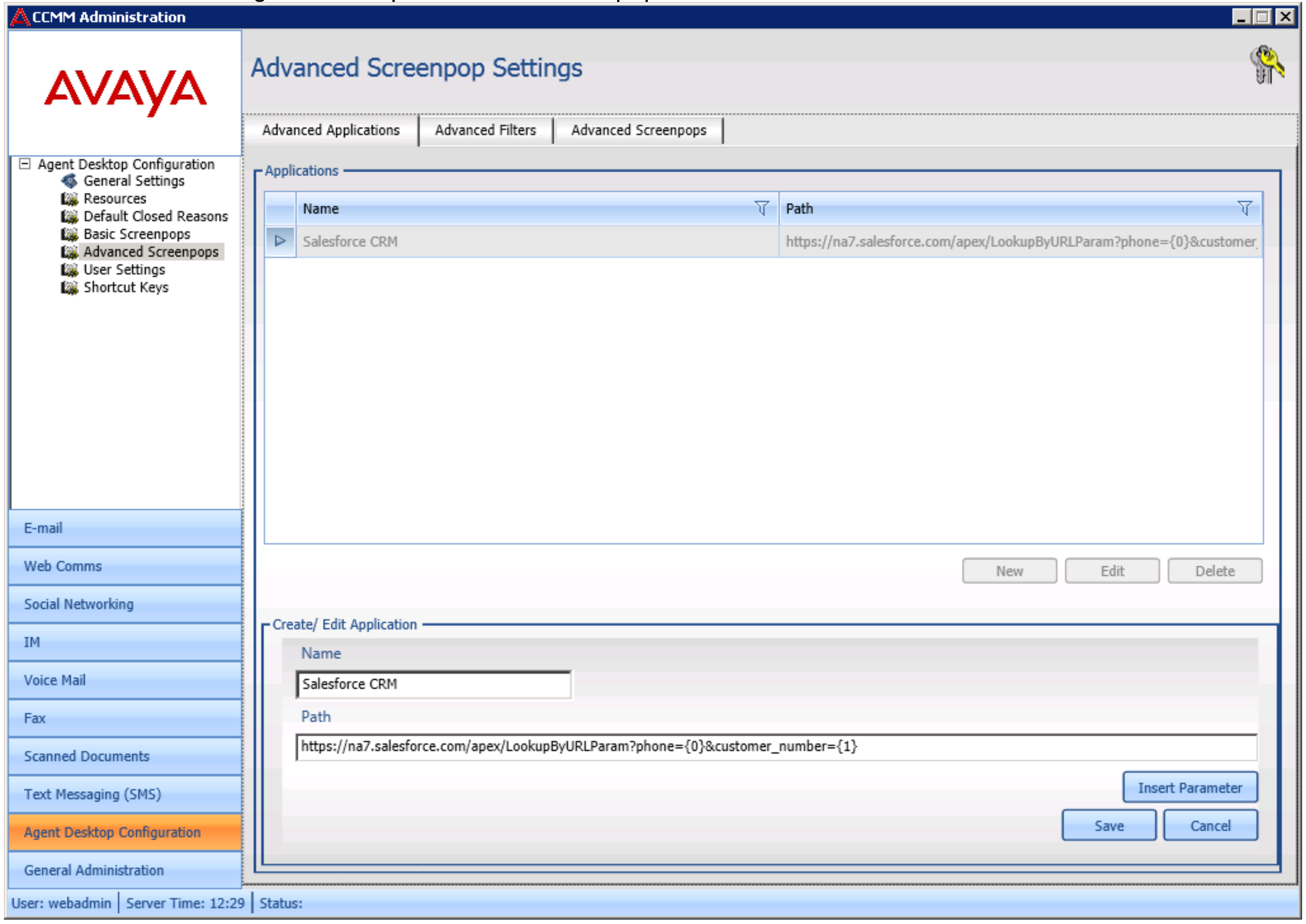
Filter screenpops by Contact Types

- [✓] Voice
- [ ] E-mail
- [ ] Web Communications
- [ ] IM
- [ ] Scanned Documents
- [ ] Fax
- [ ] Open Queue
- [ ] Outbound
- [ ] SMS
- [ ] Social Networking
- [ ] Voice Mail
- [ ] POM Outbound

4B – Screenpop on calling number and Customer number

To show how to screen pop with multiple parameters the following details screen popping on the calling number and the customer number, functionality available since Avaya Aura® Contact Center 6.4. This option provides administrators with a greater range and flexibility with respect to conditions and triggers for opening a particular screen pop. For more information, see Avaya Aura®Contact Center Server Administration (44400-610).

Within the CCMM Administration tool use the “Advanced Screenpops” option in Agent Desktop Configuration. On the first form we add an application called Salesforce CRM which defines the Path that is used later on within the Advanced Screen pop wizard. An advanced filter is optional and will be created in this example on tab two. Tab three is where the advanced screen pop wizard is launched from where we assign the newly created application and filter to the new Agent Desktop advanced screen pop of salesforce.

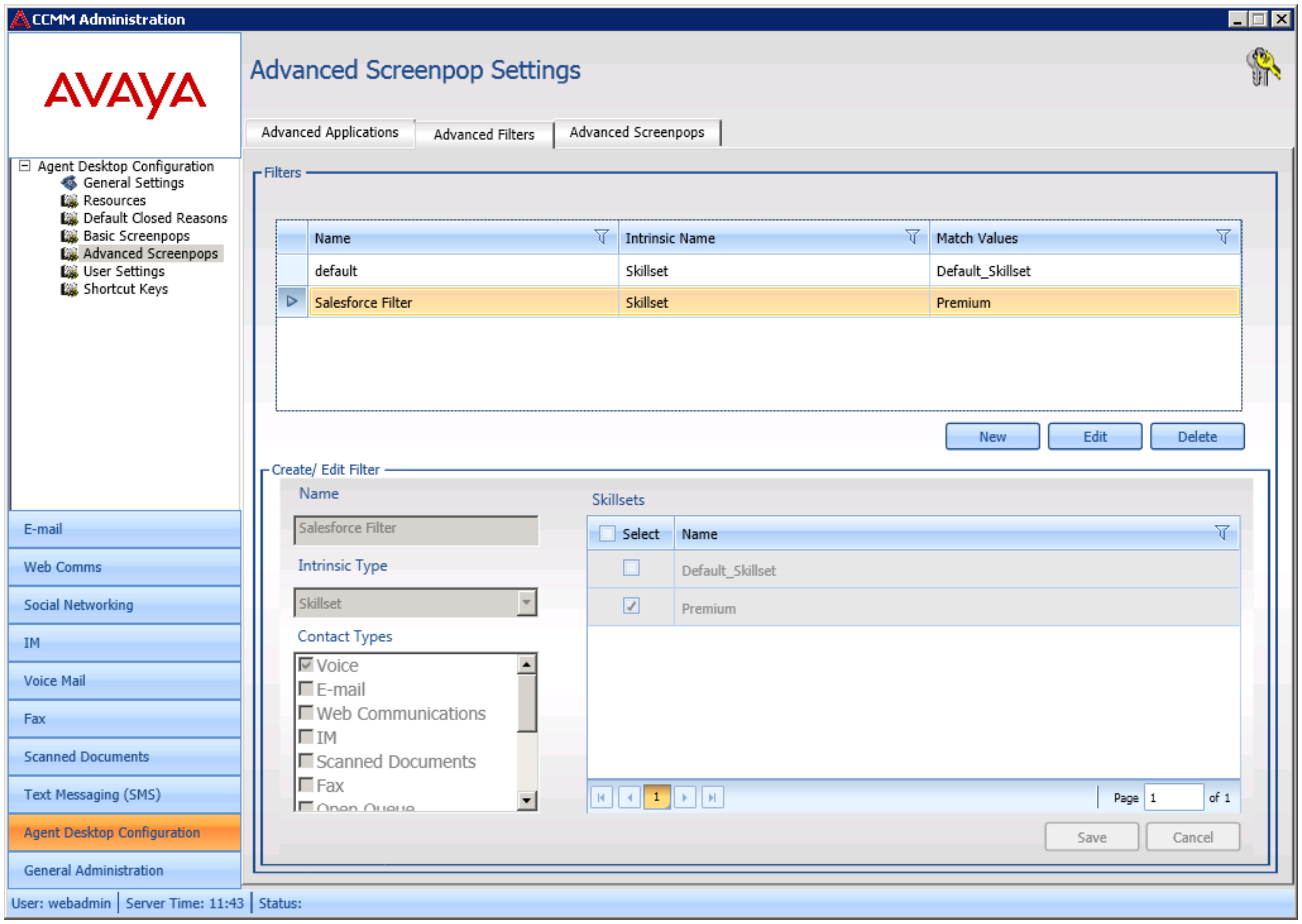


Name	Path
Salesforce CRM	https://na7.salesforce.com/apex/LookupByURLParam?phone={0}&customer

Name: Salesforce CRM

Path: https://na7.salesforce.com/apex/LookupByURLParam?phone={0}&customer_number={1}

Click the New button and add a name for the application, for example Salesforce CRM. In the path field fill in the URL to the visual force page. For each parameter we click the Insert Parameter button which adds the {0} and {1} to the URL. These will be replaced with the values of the parameters when the URL is popped at the time of a new contact in Agent Desktop. This application will be used later on during the Advanced Screenpop wizard started on that tab.



CCMM Administration

Advanced Screenpop Settings

Advanced Applications | Advanced Filters | Advanced Screenpops

Agent Desktop Configuration
- General Settings
- Resources
- Default Closed Reasons
- Basic Screenpops
- Advanced Screenpops
- User Settings
- Shortcut Keys

Filters

Name	Intrinsic Name	Match Values
default	Skillset	Default_Skillset
Salesforce Filter	Skillset	Premium

Create/ Edit Filter

Name: Salesforce Filter

Intrinsic Type: Skillset

Contact Types: [✓ Voice], E-mail, Web Communications, IM, Scanned Documents, Fax, Open Queue

Skillsets

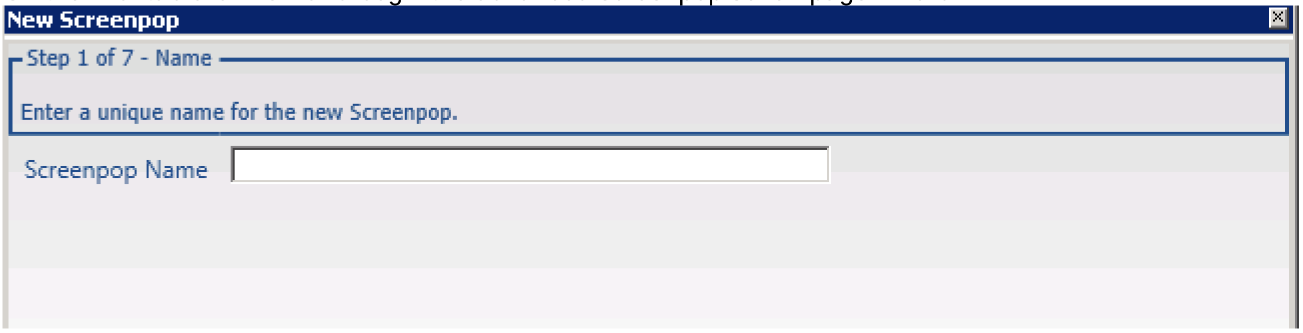
Select	Name
	Default_Skillset
✓	Premium

Page 1 of 1

User: webadmin | Server Time: 11:43 | Status:

The next tab "Advanced Filters" allows filtering of when the screenpop takes place. In this example we are going to apply a filter on the call skillset only popping for the skillset premium. Filters can be added based on contact type and also intrinsic value. The advanced filter created here is selected during the Advanced Screen pop wizard.

On the final tab click new and begin the advanced screenpop seven page wizard.



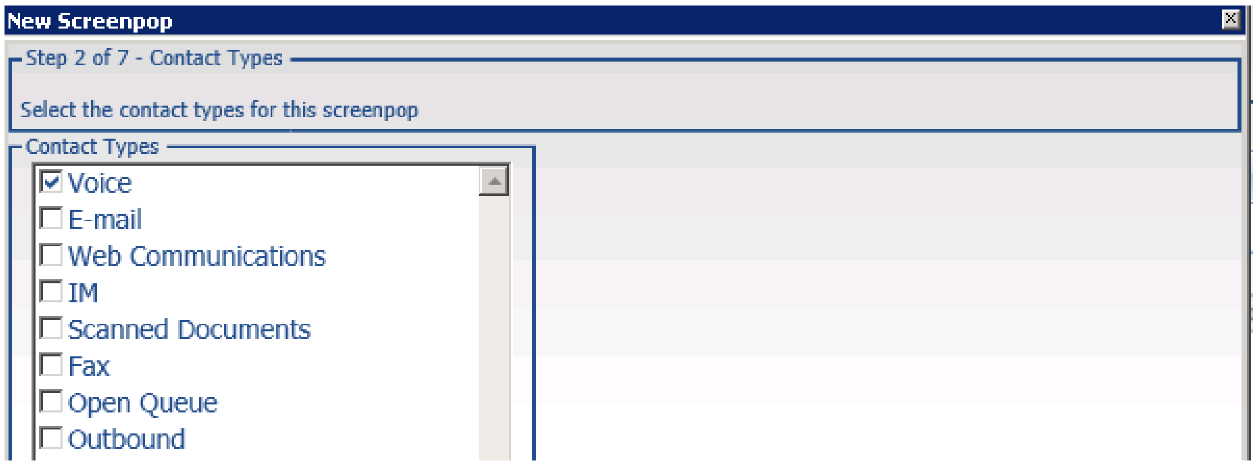
New Screenpop

Step 1 of 7 - Name

Enter a unique name for the new Screenpop.

Screenpop Name:

Step 1 enter a new name for the screen pop.



New Screenpop

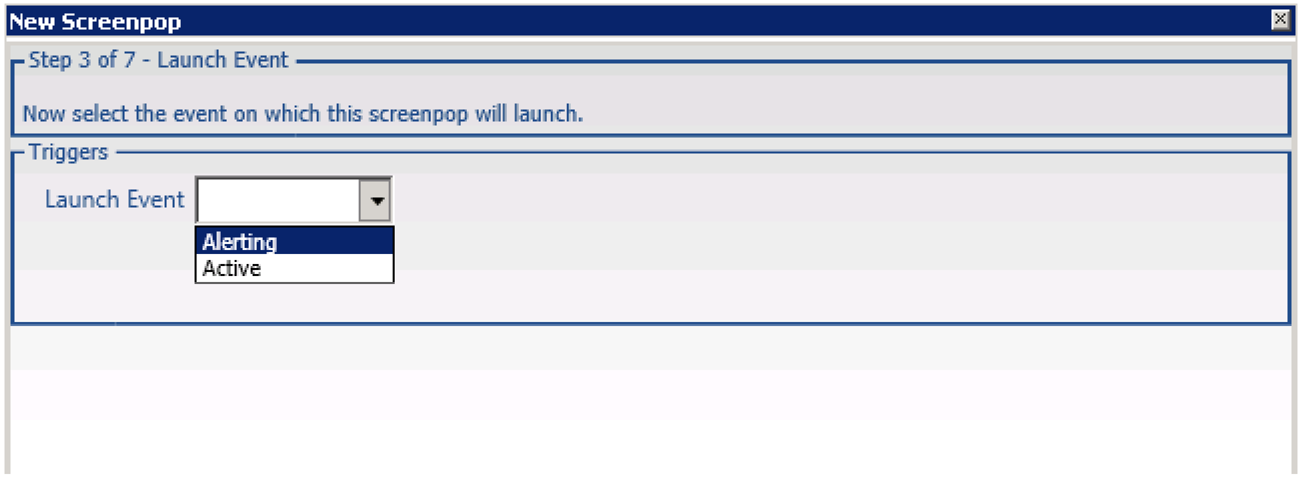
Step 2 of 7 - Contact Types

Select the contact types for this screenpop

Contact Types

- [✓ Voice]
- E-mail
- Web Communications
- IM
- Scanned Documents
- Fax
- Open Queue
- Outbound

Step 2 choose the contact type the screen pop should take place for.



New Screenpop

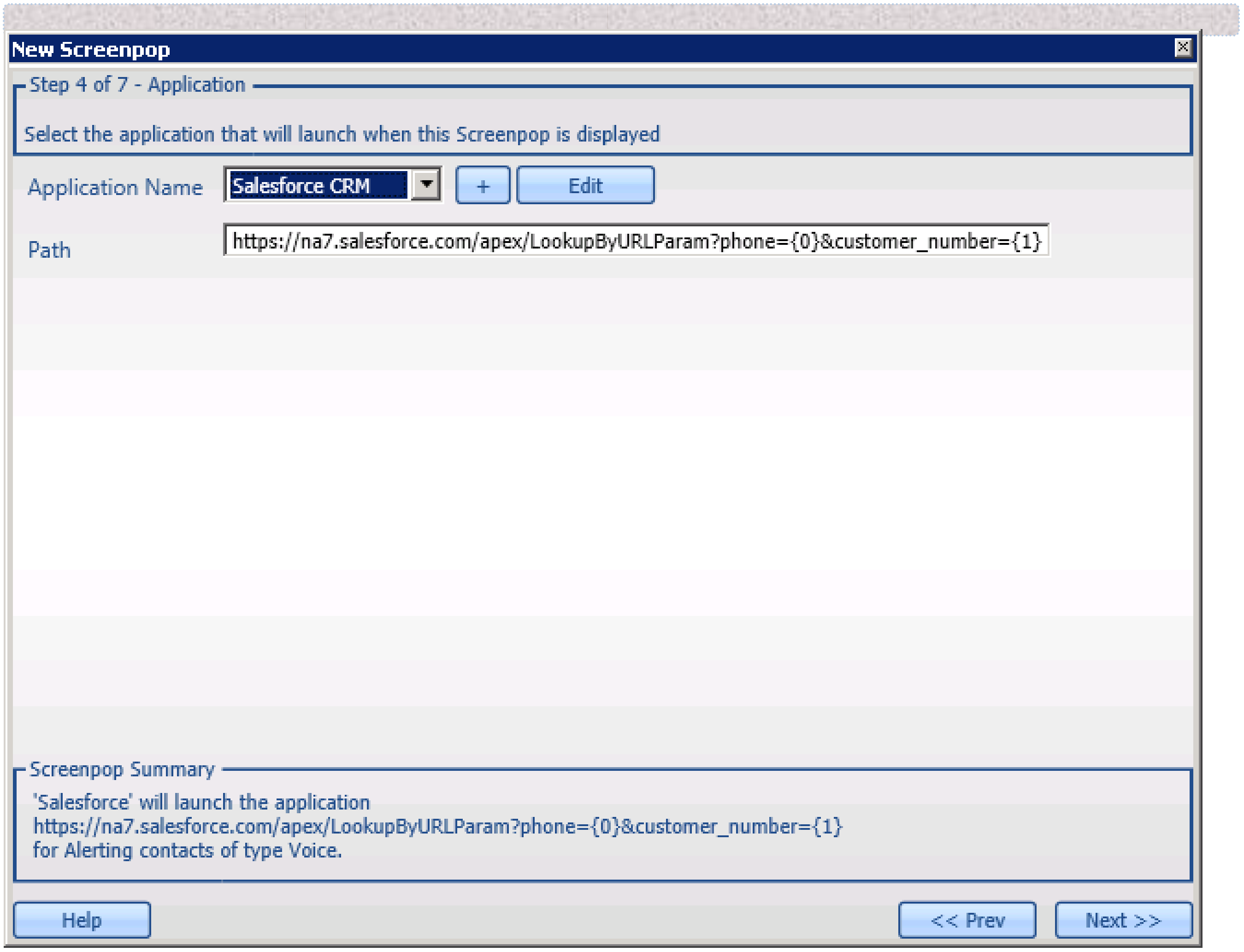
Step 3 of 7 - Launch Event

Now select the event on which this screenpop will launch.

Triggers

Launch Event

Step 3 choose whether the screenpop should take place while the call is alerting on the Agent Desktop or when it has gone active


New Screenpop

Step 4 of 7 - Application

Select the application that will launch when this Screenpop is displayed

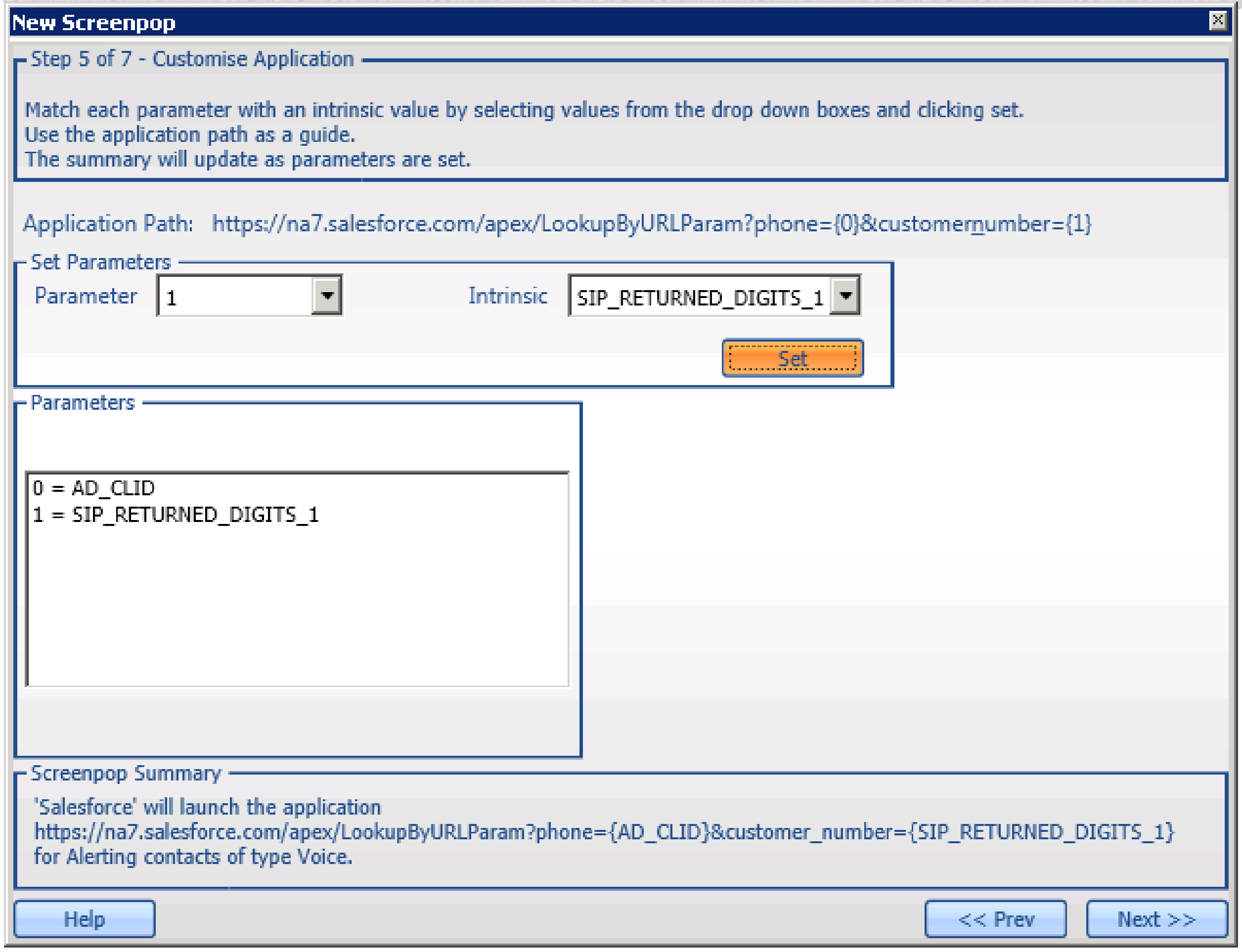
Application Name: Salesforce CRM

Path: https://na7.salesforce.com/apex/LookupByURLParam?phone={0}&customer_number={1}

Screenpop Summary

'Salesforce' will launch the application
https://na7.salesforce.com/apex/LookupByURLParam?phone={0}&customer_number={1}
for Alerting contacts of type Voice.

Steps 4 choose the application from the drop down list you created on the Advanced Applications tab earlier. This populates the path used by the screen pop.



New Screenpop

Step 5 of 7 - Customise Application

Match each parameter with an intrinsic value by selecting values from the drop down boxes and clicking set.
Use the application path as a guide.
The summary will update as parameters are set.

Application Path: https://na7.salesforce.com/apex/LookupByURLParam?phone={0}&customernumber={1}

Set Parameters

Parameter: 1

Intrinsic: SIP_RETURNED_DIGITS_1

Parameters

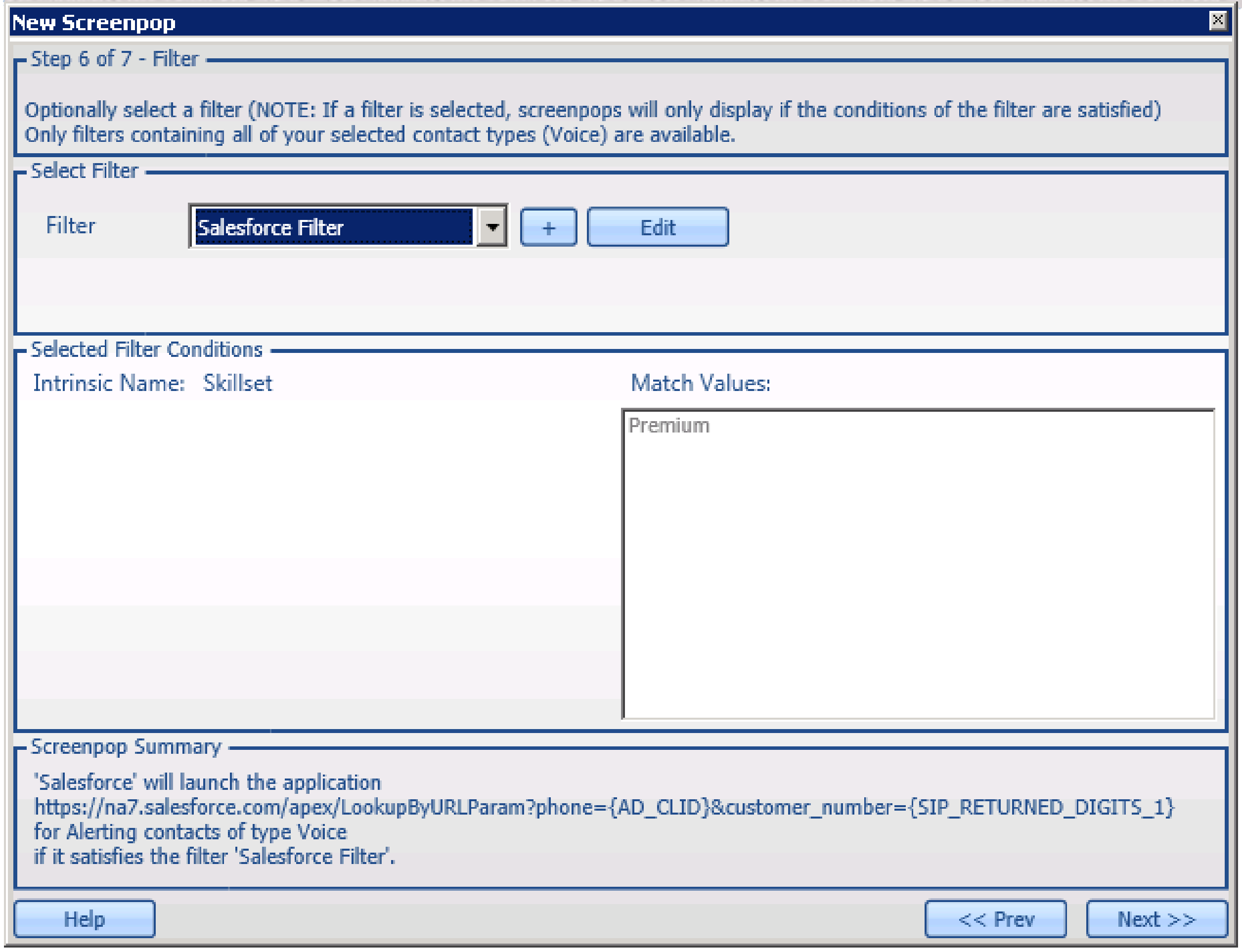
0 = AD_CLID
1 = SIP_RETURNED_DIGITS_1

Screenpop Summary

'Salesforce' will launch the application
https://na7.salesforce.com/apex/LookupByURLParam?phone={AD_CLID}&customer_number={SIP_RETURNED_DIGITS_1}
for Alerting contacts of type Voice.

Step 5 Each parameter in the application path is set with an intrinsic value. Choose the parameter from the drop down list and the matching intrinsic and click Set. This then adds the entries in the Parameters box. SIP_Returned_Digits_1 stores the digits added during an IVR session. In this example the IVR session collected a customer number from the end customer which will be used in the salesforce lookup to determine the correct account to pop for the agent.

Important to note: Populated intrinsics vary depending on contact type and switch type.



Step 6 of 7 - Filter

Optionally select a filter (NOTE: If a filter is selected, screenpops will only display if the conditions of the filter are satisfied)
Only filters containing all of your selected contact types (Voice) are available.

Select Filter

Filter: Salesforce Filter

Selected Filter Conditions

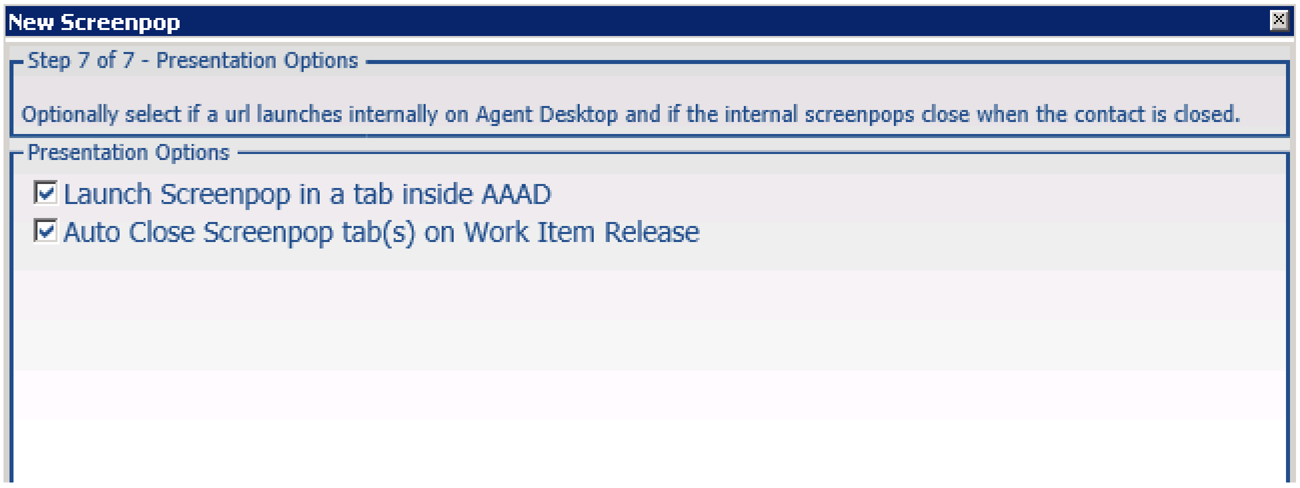
Intrinsic Name: Skillset

Match Values: Premium

Screenpop Summary

'Salesforce' will launch the application
https://na7.salesforce.com/apex/LookupByURLParam?phone={AD_CLID}&customer_number={SIP_RETURNED_DIGITS_1}
for Alerting contacts of type Voice
if it satisfies the filter 'Salesforce Filter'.

Step 6 from the drop down list choose the filter to be applied to the screenpop created earlier on the Advanced Filters tab. A filter is optional.
Note the “Screenpop Summary” box at the end of the step 6 form. This in plain English summarises what will occur when the filters (which are optional) are satisfied.



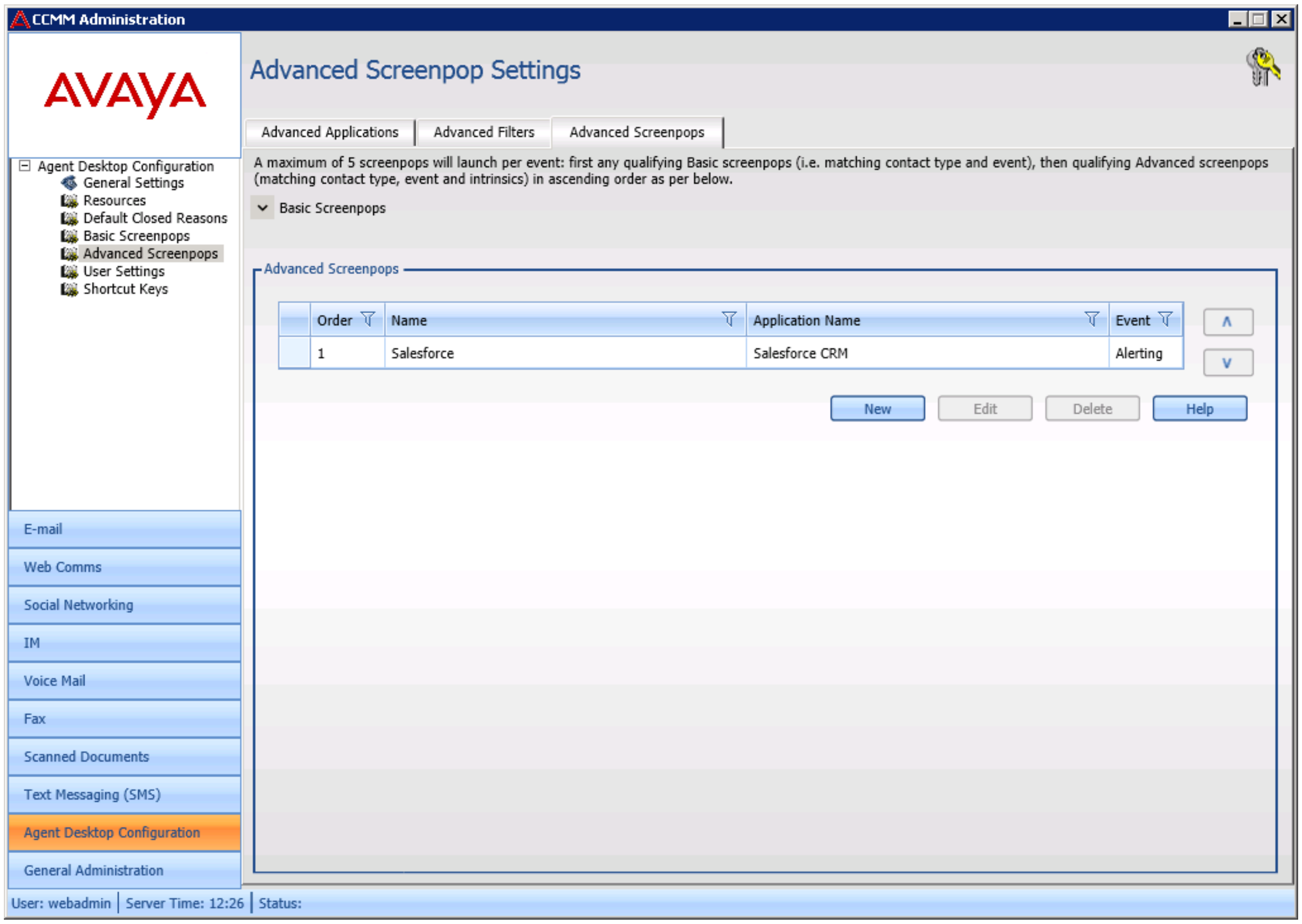
Step 7 of 7 - Presentation Options

Optionally select if a url launches internally on Agent Desktop and if the internal screenpops close when the contact is closed.

Presentation Options

☑ Launch Screenpop in a tab inside AAAD
☑ Auto Close Screenpop tab(s) on Work Item Release

Step 7 the final step allows setting whether or not the screen pop launches within the Agent Desktop application or outside it in an independent window. This step also sets on or off the auto close of the screen pop on the release of a work item.



Advanced Screenpop Settings

A maximum of 5 screenpops will launch per event: first any qualifying Basic screenpops (i.e. matching contact type and event), then qualifying Advanced screenpops (matching contact type, event and intrinsics) in ascending order as per below.

Advanced Screenpops

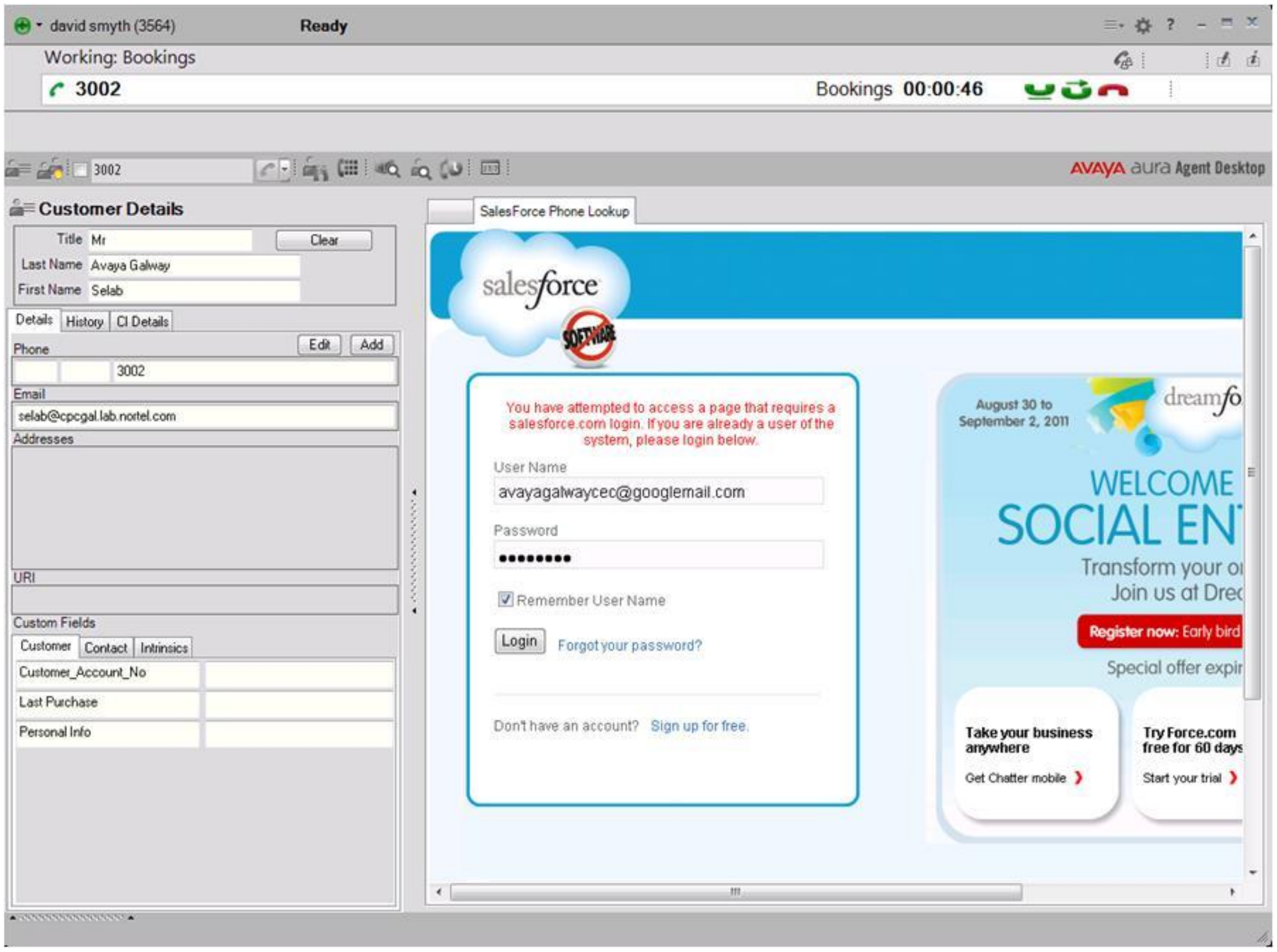
Order	Name	Application Name	Event
1	Salesforce	Salesforce CRM	Alerting

User: webadmin | Server Time: 12:26 | Status:

Adding the advanced screen pop is now complete and can be viewed in the Advanced Screenpops list in the CCMM Administration application.

Testing of the new settings involve placing in a call to the appropriate skillset and once the agent answers the call, then they should receive a pop of Salesforce. The first call into the agent will require the agent to login to Salesforce using supplied a Salesforce username and password. Each Enterprise will be different. Some sites may have a generic Salesforce login for every agent or each agent may have a unique login.

No matter what method is adopted by the Enterprise, the agent will perform a onetime authentication to Salesforce in order to retrieve any customer details. See following screen grab for a typical first time login example.



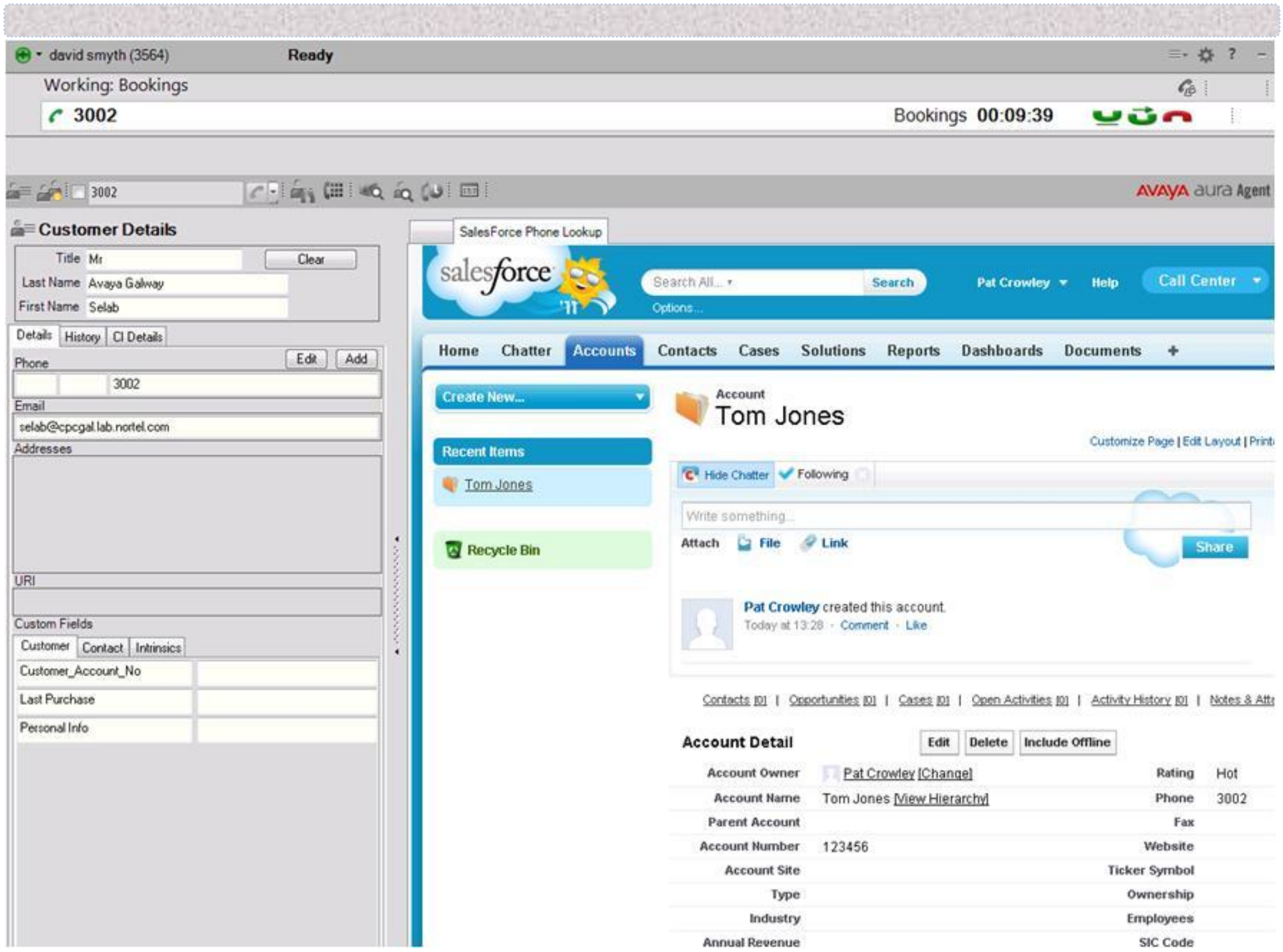
Title	Mr
Last Name	Avaya Galway
First Name	Selab
Phone	3002
Email	selab@cpcgal.lab.nortel.com

You have attempted to access a page that requires a salesforce.com login. If you are already a user of the system, please login below.

User Name: avayagalwaycec@googlemail.com

Password: ●●●●●●●●●

At this point, the agent will enter an Enterprise provided Salesforce login account name and password. As stated previously, this can be unique to every agent or it can be a global enterprise account. Once authentication is successful, then the following customer account matching telephone number 3002 will be displayed within Agent Desktop. Note that there are multiple Agent Desktop settings which define when it occurs (on ringing or on answer) and whether it is inside or outside the AAAD frame.



Working: Bookings
3002 — Bookings 00:09:39

Title	Mr
Last Name	Avaya Galway
First Name	Selab
Phone	3002
Email	selab@cpcgal.lab.nortel.com
Addresses	
URI	
Customer_Account_No	
Last Purchase	
Personal Info	

Account
Tom Jones

Pat Crowley created this account.
Today at 13:28 · Comment · Like

Account Detail

Account Owner	Pat Crowley [Change]	Rating	Hot
Account Name	Tom Jones [View Hierarchy]	Phone	3002
Parent Account		Fax	
Account Number	123456	Website	
Account Site		Ticker Symbol	
Type		Ownership	
Industry		Employees	
Annual Revenue		SIC Code	

In the above example, an incoming call from extension 3002 matches the account Tom Jones and that is what is displayed in the Agent Desktop application. Hence this is the out of the box simple Avaya Contact Center screen pop.

Appendix 5 includes the sample Apex code for delivering the ability to screen pop customer details based on email address.

Once plug-in installation and Agent Desktop configuration are complete, as outlined in this document, the following process will be possible.

- Agent Desktop receives an incoming call with calling line identification or customer account number (e.g. collected through an IVR menu).
- Agent Desktop sends a request to the Avaya Aura Agent Desktop Salesforce Plug-in. including the calling line identification or customer account number or both.
- The Avaya Aura Agent Desktop Salesforce Plug-in performs an Account look up on salesforce.com using the phone number or the customer number or both.

- The Avaya Aura Agent Desktop CRM Plug-in redirects the agent’s browser to the relevant Account on salesforce.com.

This concludes the first part of this white paper – client side integration. For the next half of the document we will describe optional but more complex integration which involve server side integration between Avaya Contact Center (CCMS Database Integration Wizard) and Salesforce

PART II

4 – AVAYA CONTACT CENTER SERVER SIDE CONFIGURATION

Part I of this document describes show to enable basic screen pop capabilities. Part II now describes how to configure Avaya Contact Center to perform an additional lookup of Salesforce for additional customer information (customer priority, preferred agent, etc) while the contact is waiting to be assigned to an agent in an appropriate skillset. It is important to complete all of the following chapters in the sequence they appear in this document.

As briefly mentioned in the opening chapter there are four key pieces of work to be completed:- (note steps required for client side must be completed before server side can be made fully operational)

1. Enabling the CCMS component to securely logon to Salesforce CRM using pre-defined parameters. This is required if we want to make a “routing” decision based on information in Salesforce.
2. Developing new code modules within the Salesforce API’s (known as APEX) which will be used to retrieve the “agent ID” or “Customer Priority” and pass it back to Avaya Contact Center.
3. Defining Orchestration Designer workflow which will assign the customer contact to the preferred agent with the correct agent ID.
4. Configuring the screen pop parameters using CCMM administration application – Client side integration already described in part 1.

We will now describe the process and procedures to complete steps 1, 2 & 3. Step 4 is already complete.

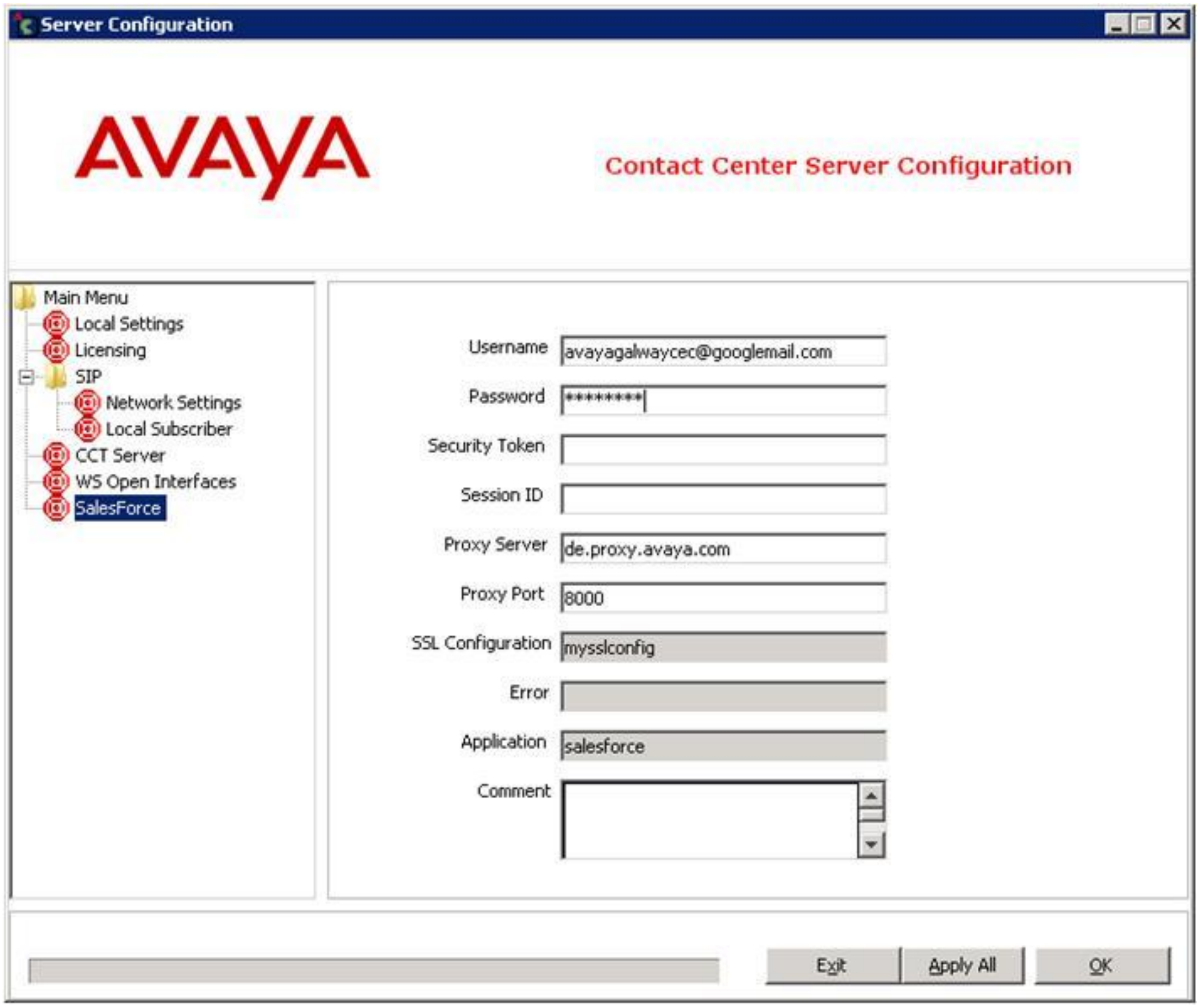
CCMS Security Token Configuration using Salesforce Plug-in

Just as in the case of the client integration, we have to enable the Avaya Contact Center server to securely connect and exchange information with Salesforce. To do this we need to enable security tokens and session ID’s as follows.

With the release of Avaya Contact Center 6.1 in November 2010, Avaya Contact Center now has a Salesforce configuration element added to the CCMS server configuration application. With Avaya Contact Center 6.2 this is further enhanced to take into account Enterprise use of web proxy’s for re-directing external web traffic including web services.

For Avaya Aura Contact Center (specifically CCMS) to communicate with web services being exposed by Salesforce it needs to retrieve a *sessionID* from Salesforce. This *sessionID* is a unique ID associated with this session and needs to be set in the header of every SOAP message issued to Salesforce. On successful completion of the login process a *sessionID* is returned. This sessionID is only valid for a period of time and will typically timeout after 2 hours (120 min) of inactivity. This timeout is automated and controlled by Salesforce. If the sessionID times out, a new sessionID is retrieved from Salesforce. Please refer to the Salesforce documentation for further details.

In order to program the Salesforce connector on CCMS, the CCMS server configuration element must be accessed by the system administrator. To access server configuration, on CCMS, go to Start/Programs/Avaya/Contact Center Manager Server/Server Configuration. Once open, click on the “Salesforce” plug-in tab as follows:



Username	avayagalwaycec@googlemail.com
Password	*********
Security Token	
Session ID	
Proxy Server	de.proxy.avaya.com
Proxy Port	8000
SSL Configuration	mysslconfig
Error	
Application	salesforce
Comment	

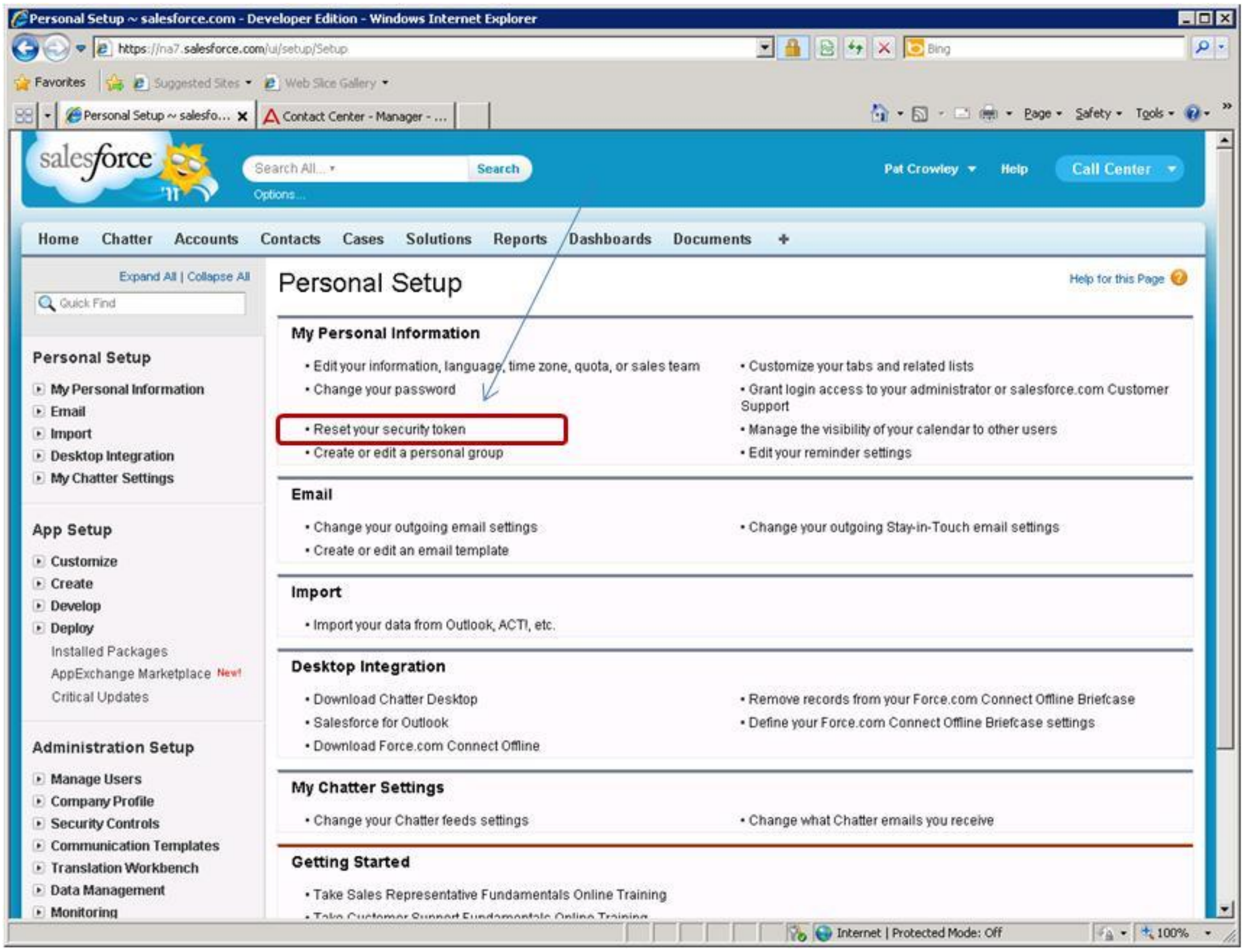
Note: The Salesforce Tab shown above is only available on Avaya Contact Center 6.1 SP2 systems onwards with the addition of the proxy fields available from release Avaya Contact Center 6.2 onwards.

The Salesforce dialog contained within the Server Configuration on the CCMS server allows a system administrator to enter the necessary login details. When an information request is made to a Salesforce service from Contact Center a sessionID is retrieved and stored locally to be used in subsequent calls. The following table contains the data that needs to be entered at configuration time before any information is transferred between Avaya Contact Center and Salesforce.

Name	Value	Description
Username	Valid Salesforce user A/C	
Password	Password associated with user	
Security Token	Authentication of User	This security token is generated from the Salesforce site (described previously and is appended onto the Password.

It is mandatory to enter the above three parameters into CCMS Salesforce configuration tab. Proxy Server and port are optional and depend on your enterprise web hosting requirements.

At this stage of the process, you cannot complete the CCMS configuration as you do not have any value for the Security Token. This needs to be generated while logged in on your Salesforce developer account. Login to the account, select “My Profile” from drop down under your login name to arrive at the following window.



Personal Setup ~ salesforce.com - Developer Edition - Windows Internet Explorer

https://na7.salesforce.com/ui/setup/Setup

Home Chatter Accounts Contacts Cases Solutions Reports Dashboards Documents

Personal Setup

Personal Setup

- My Personal Information
- Email
- Import
- Desktop Integration
- My Chatter Settings

App Setup

- Customize
- Create
- Develop
- Deploy
- Installed Packages
- AppExchange Marketplace New!
- Critical Updates

Administration Setup

- Manage Users
- Company Profile
- Security Controls
- Communication Templates
- Translation Workbench
- Data Management
- Monitoring

My Personal Information

- Edit your information, language, time zone, quota, or sales team
- Change your password
- Reset your security token
- Create or edit a personal group
- Customize your tabs and related lists
- Grant login access to your administrator or salesforce.com Customer Support
- Manage the visibility of your calendar to other users
- Edit your reminder settings

Email

- Change your outgoing email settings
- Create or edit an email template
- Change your outgoing Stay-in-Touch email settings

Import

- Import your data from Outlook, ACT!, etc.

Desktop Integration

- Download Chatter Desktop
- Salesforce for Outlook
- Download Force.com Connect Offline
- Remove records from your Force.com Connect Offline Briefcase
- Define your Force.com Connect Offline Briefcase settings

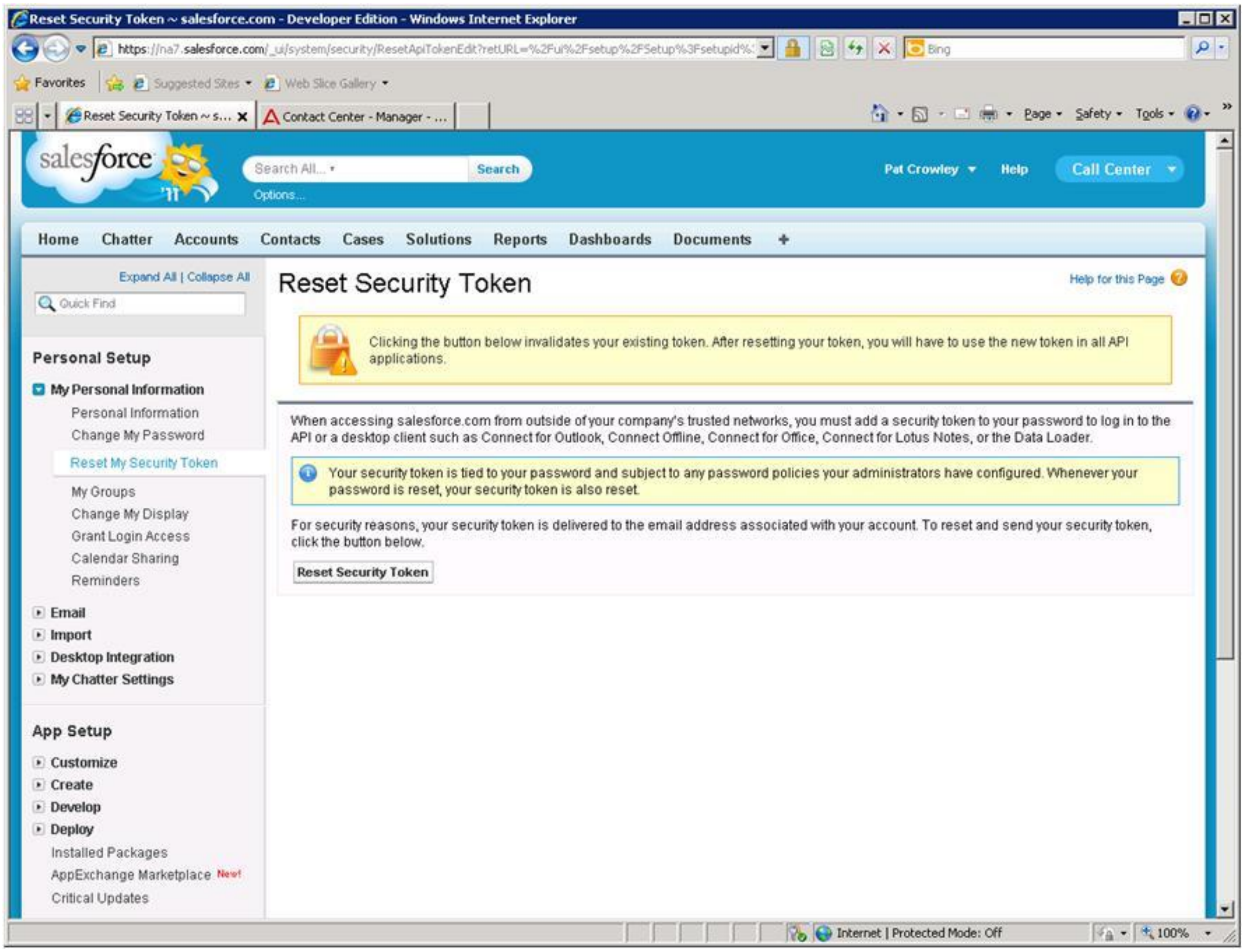
My Chatter Settings

- Change your Chatter feeds settings
- Change what Chatter emails you receive

Getting Started

- Take Sales Representative Fundamentals Online Training

Now click on "Reset your security token" to arrive here.



Reset Security Token

Clicking the button below invalidates your existing token. After resetting your token, you will have to use the new token in all API applications.

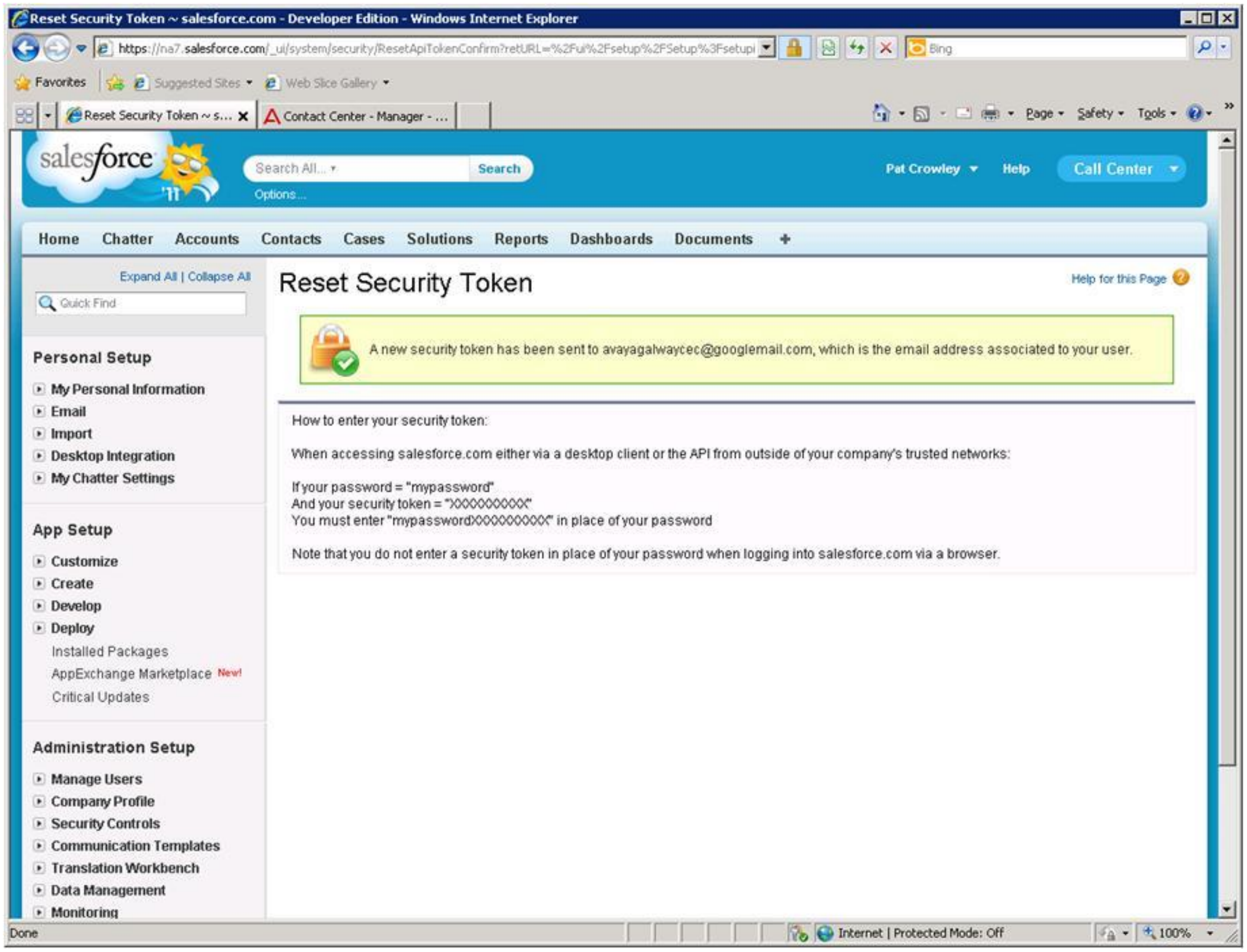
When accessing salesforce.com from outside of your company's trusted networks, you must add a security token to your password to log in to the API or a desktop client such as Connect for Outlook, Connect Offline, Connect for Office, Connect for Lotus Notes, or the Data Loader.

Your security token is tied to your password and subject to any password policies your administrators have configured. Whenever your password is reset, your security token is also reset.

For security reasons, your security token is delivered to the email address associated with your account. To reset and send your security token, click the button below.

Reset Security Token

Now click on "**Reset Security Token**" taking note of the various warnings and text above. This action will generate an email to your registered Salesforce email account as described here.


Reset Security Token

A new security token has been sent to avayagalwaycec@googlemail.com, which is the email address associated to your user.

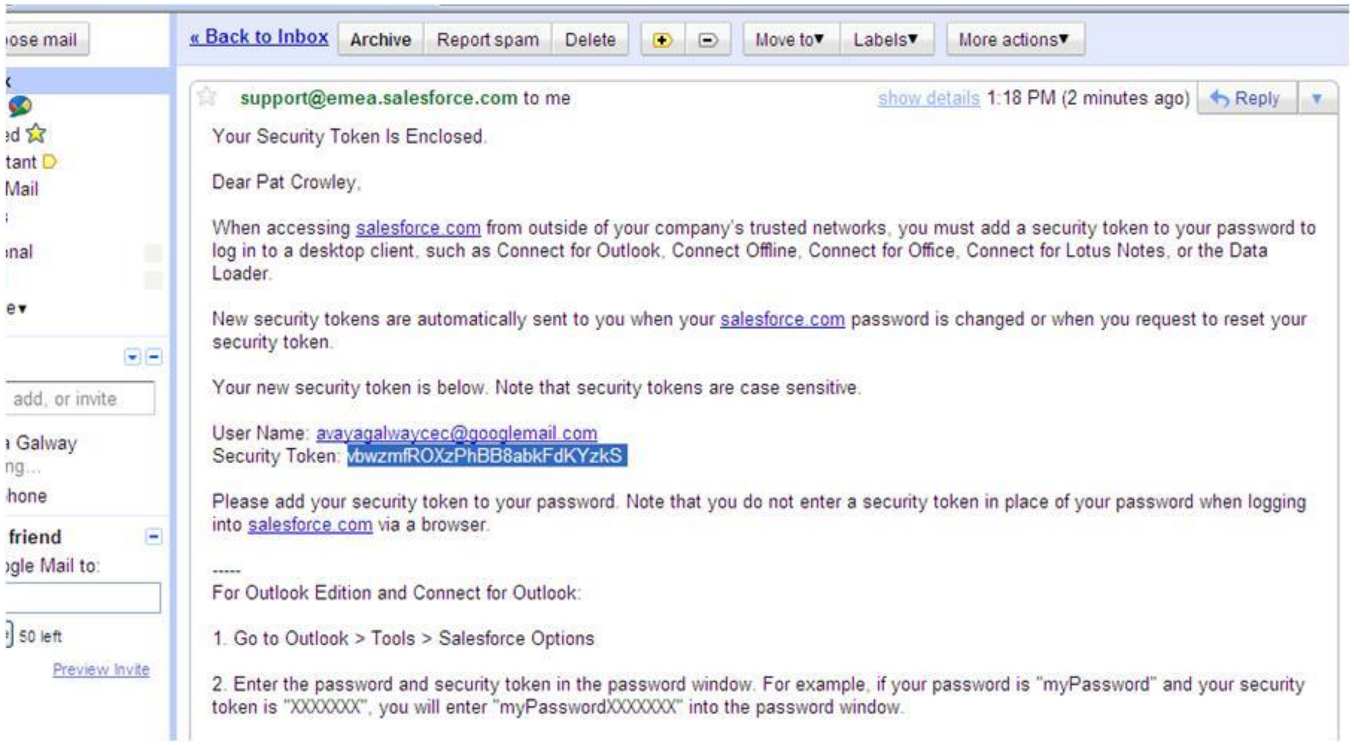
How to enter your security token:

When accessing salesforce.com either via a desktop client or the API from outside of your company's trusted networks:

If your password = "mypassword"
And your security token = "XXXXXXXXXX"
You must enter "mypasswordXXXXXXXXXX" in place of your password

Note that you do not enter a security token in place of your password when logging into salesforce.com via a browser.

In your email account associated with your Salesforce account, you should now be seeing an email with the new security token as follows



support@emea.salesforce.com to me 1:18 PM (2 minutes ago)

Your Security Token Is Enclosed.

Dear Pat Crowley,

When accessing salesforce.com from outside of your company's trusted networks, you must add a security token to your password to log in to a desktop client, such as Connect for Outlook, Connect Offline, Connect for Office, Connect for Lotus Notes, or the Data Loader.

New security tokens are automatically sent to you when your salesforce.com password is changed or when you request to reset your security token.

Your new security token is below. Note that security tokens are case sensitive.

User Name: avayagalwaycec@googlemail.com
Security Token: vbwzmfROXzPhBB8abkFdKYzkS

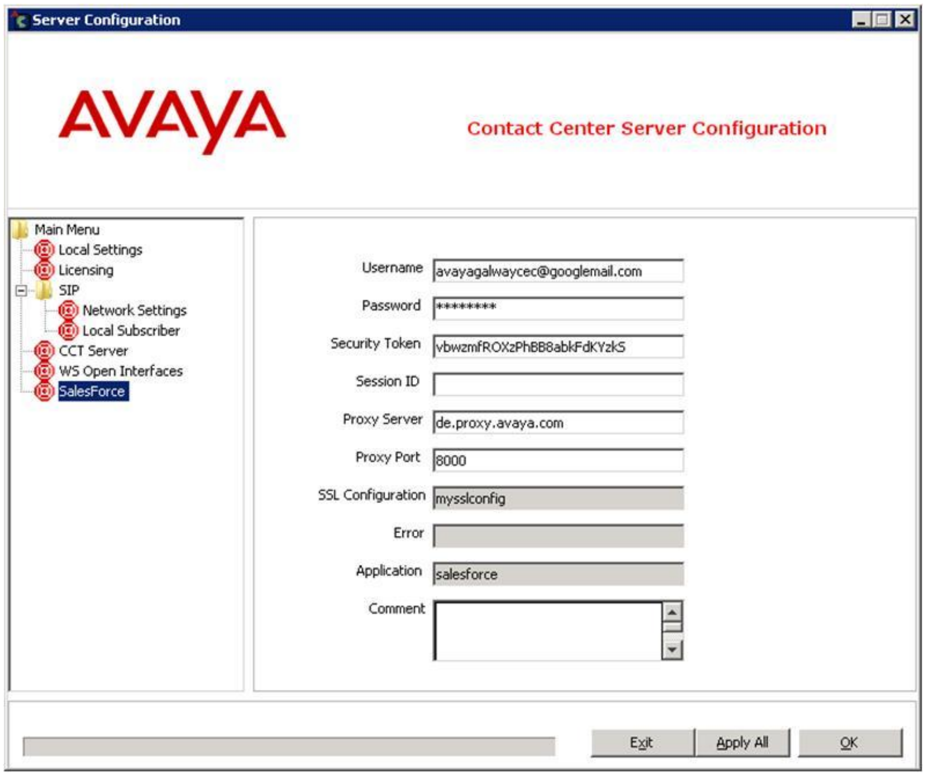
Please add your security token to your password. Note that you do not enter a security token in place of your password when logging into salesforce.com via a browser.

For Outlook Edition and Connect for Outlook:

1. Go to Outlook > Tools > Salesforce Options

2. Enter the password and security token in the password window. For example, if your password is "myPassword" and your security token is "XXXXXXX", you will enter "myPasswordXXXXXXX" into the password window.

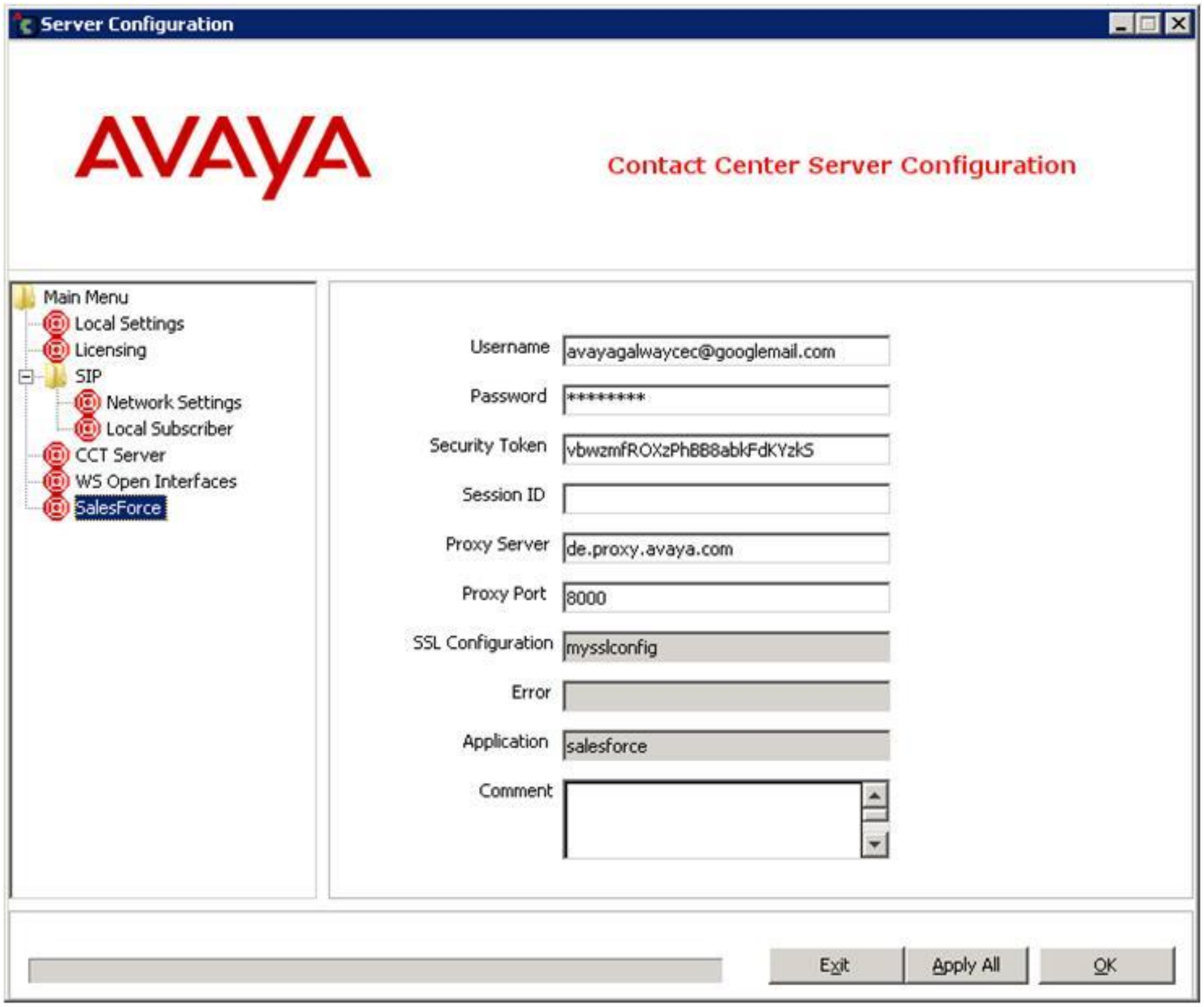
Now back on your Avaya Contact Center (CCMS server configuration) manually enter the new security token as follows



Username: avayagalwaycec@googlemail.com
Password: ********
Security Token: vbwzmfROXzPhBB8abkFdKYzkS
Session ID:
Proxy Server: de.proxy.avaya.com
Proxy Port: 8000
SSL Configuration: mysslconfig
Error:
Application: salesforce
Comment:

When entered, click “Apply All”. When the process has completed (30 seconds), you will be prompted to reboot the Avaya Contact Center server. This is mandatory but in production environments, this will have to be performed out of hours. If you wish to restart later, and then click No otherwise proceed with the restart. If NO, then click on exit to leave the server configuration application. Post CCMS restart, the system administrator can check the settings once again in the Salesforce configuration tab.

Note - We will not have a valid connection from CCMS into Salesforce until we actually build our workflow and perform a test web service connection. The session ID field below will still be empty until at least TWO test web service calls are made into Salesforce. We will describe that procedure in chapter 6.



Username	avayagalwaycec@googlemail.com
Password	*********
Security Token	vbwzmfROXzPhBB8abkFdKYzkS
Session ID	
Proxy Server	de.proxy.avaya.com
Proxy Port	8000
SSL Configuration	mysslconfig
Error	
Application	salesforce
Comment	

The following values are automatically updated within CCMS once a valid connection is established between CCMS and Salesforce. We need to complete the steps in Chapters 5 & 6 before we see any values in these fields.

-- -----

SessionID (read only) This value reflects the current sessionID that is being used by Avaya Contact Center to contact Salesforce web services.

SSL Configuration

Error (read only) If an exception occurs when trying to login into Salesforce the message will be displayed here.

Comment (read only) Will detail information when a sessionID has be retrieved or destroyed.

----

5 – DEEP DIVE SALESFORCE APEX WEB SERVICES

Salesforce CRM is an OPEN, scalable CRM system with open interfaces and SDK's available to programmers to develop their own applications in Salesforce. Avaya Contact Center utilises the "Apex" web services to provide the integration between the two platforms.

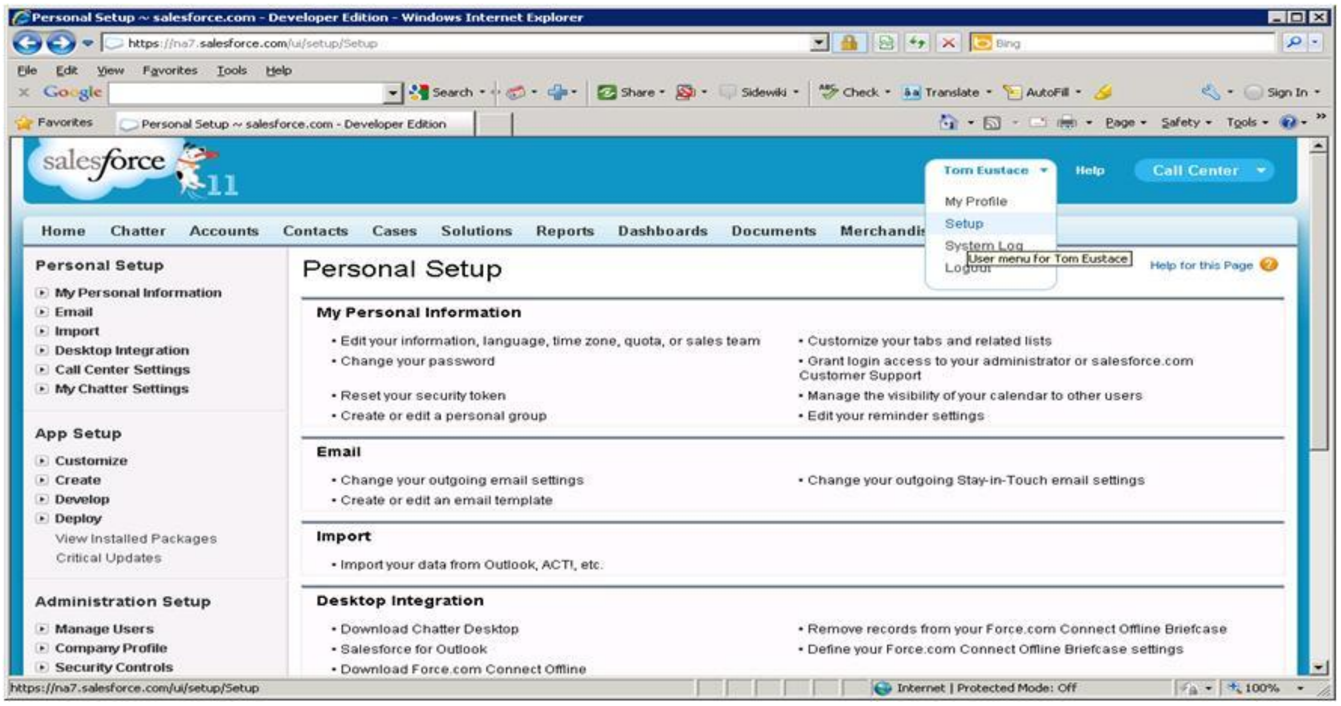
Apex Web Services allow programmers to write Apex logic and expose the logic as a web service that an external application can invoke. This allows the programmer expose their Apex code/scripts so that external applications (like Avaya Contact Center) can access the Apex code. In other words, programmers can "program" Salesforce with their own code within their own customer account and allow external applications interact with this code using web service methodology to exchange information.

Avaya has registered with Salesforce for a user account with developer access. With this account, Avaya can now write its own code modules within Salesforce to provide information to Avaya applications such as Avaya Contact Center. The account used by Avaya is a standard Salesforce user account with developer capabilities.

Developing Apex Code

To access the Apex API, a programmer must first login using their Salesforce username and password. Then under the account name, select the drop down arrow and in the menu option select "Setup". See following image.

avaya.com



Personal Setup

My Personal Information

- Edit your information, language, time zone, quota, or sales team
- Change your password
- Reset your security token
- Create or edit a personal group
- Customize your tabs and related lists
- Grant login access to your administrator or salesforce.com Customer Support
- Manage the visibility of your calendar to other users
- Edit your reminder settings

Email

- Change your outgoing email settings
- Create or edit an email template
- Change your outgoing Stay-in-Touch email settings

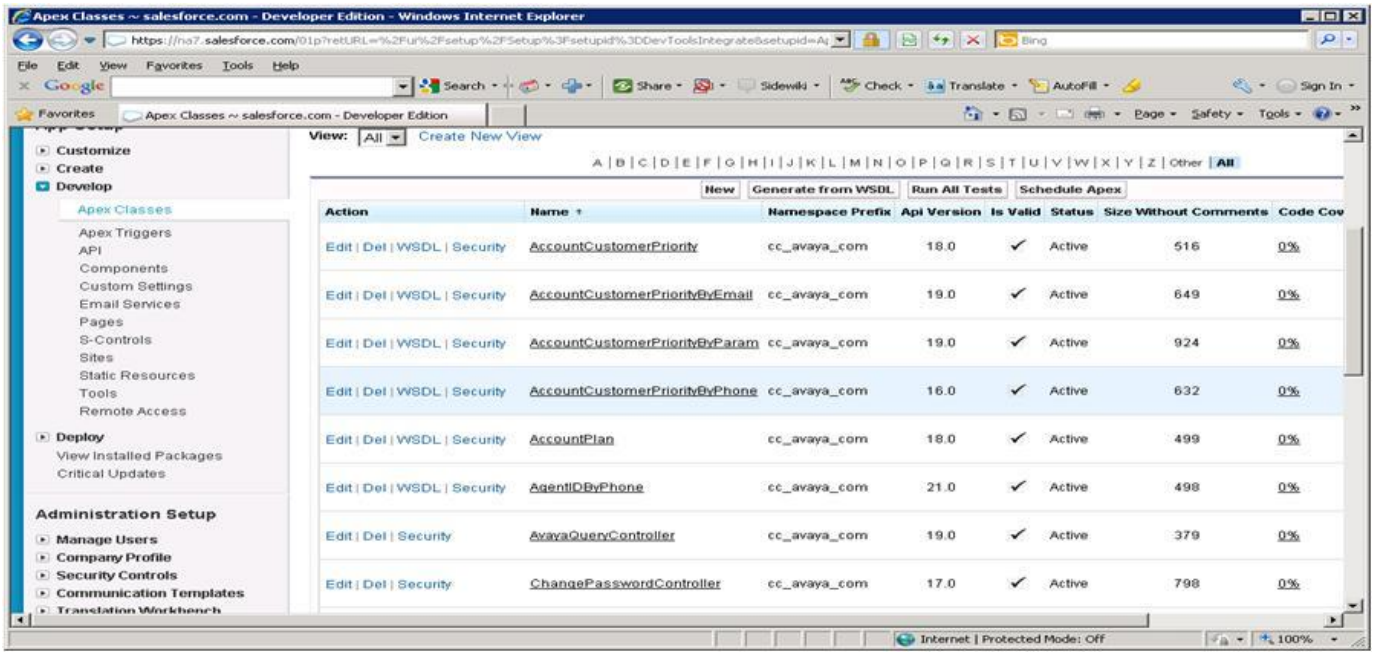
Import

- Import your data from Outlook, ACT!, etc.

Desktop Integration

- Download Chatter Desktop
- Salesforce for Outlook
- Download Force.com Connect Offline
- Remove records from your Force.com Connect Offline Briefcase
- Define your Force.com Connect Offline Briefcase settings

You will then see the following menu. Select the “Develop” option followed by “Apex Classes”.



Action	Name ↑	Namespace Prefix	Api Version	Is Valid	Status	Size Without Comments	Code Cov
Edit | Del | WSDL | Security	AccountCustomerPriority	cc_avaya_com	18.0	✓	Active	516	0%
Edit | Del | WSDL | Security	AccountCustomerPriorityByEmail	cc_avaya_com	19.0	✓	Active	649	0%
Edit | Del | WSDL | Security	AccountCustomerPriorityByParam	cc_avaya_com	19.0	✓	Active	924	0%
Edit | Del | WSDL | Security	AccountCustomerPriorityByPhone	cc_avaya_com	16.0	✓	Active	632	0%
Edit | Del | WSDL | Security	AccountPlan	cc_avaya_com	18.0	✓	Active	499	0%
Edit | Del | WSDL | Security	AgentIDByPhone	cc_avaya_com	21.0	✓	Active	498	0%
Edit | Del | Security	AvayaQueryController	cc_avaya_com	19.0	✓	Active	379	0%
Edit | Del | Security	ChangePasswordController	cc_avaya_com	17.0	✓	Active	798	0%

The above image is taken from the Avaya user account on Salesforce and not the new account we created earlier.

Avaya have created and developed a number of different code modules. Every enterprise customer wishing to develop integration with Salesforce will have to develop their own set of Apex classes or code that is particular to their own account. The above examples from the Avaya account are not available currently available to other account holders in Salesforce but other account holders are free to develop them using examples given in this white paper. Other examples of Apex classes are given later in this document including Lookup by Email which allows a lookup to be performed based on customers email address. Alternatively Avaya and its Professional Services Organisations can deliver the same applications and code modules for enterprise customers.

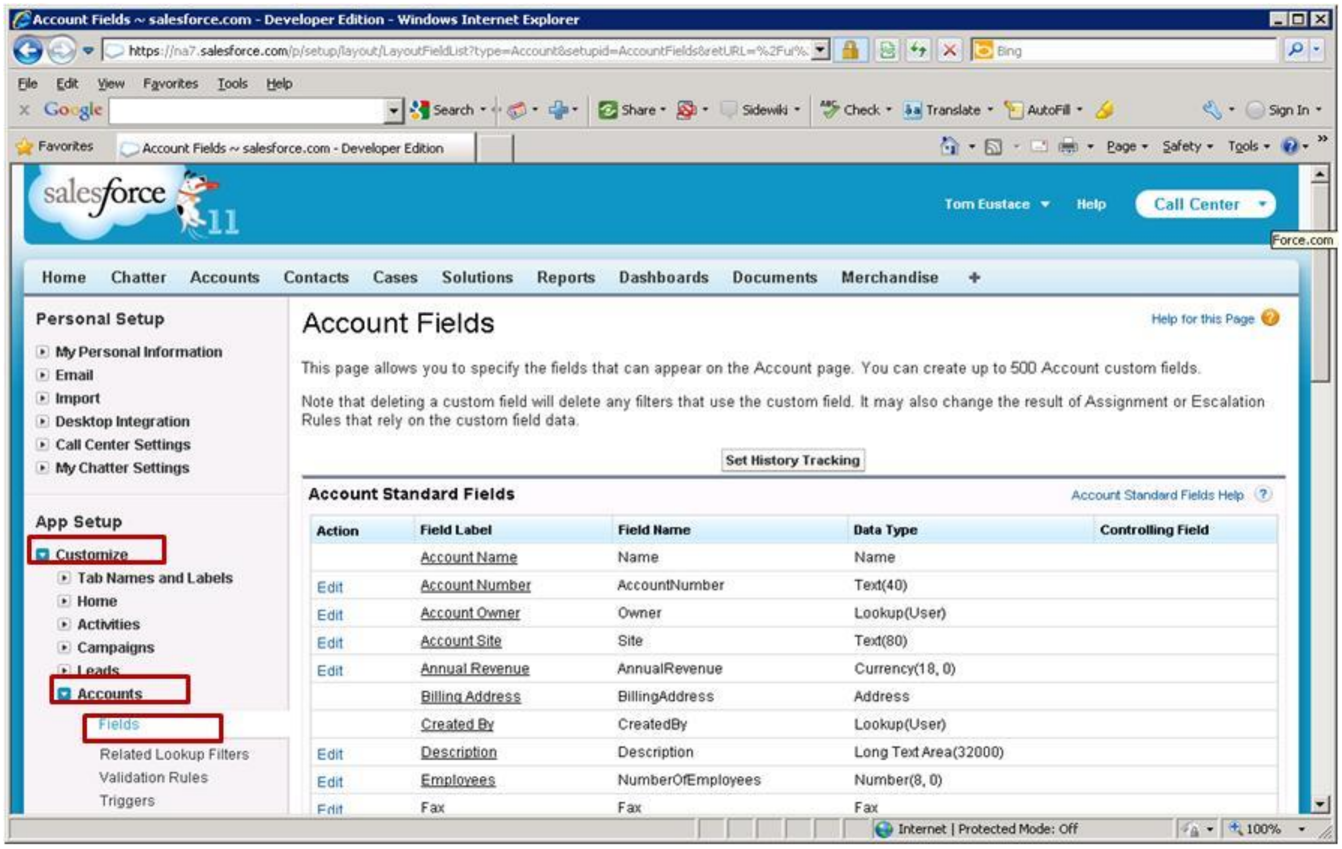
Some of the examples already developed by Avaya allow external applications to search Salesforce for customer account details based on customer phone record, customer address, customer email address, customer account priority etc. For example if we wish to use the customer phone number to search Salesforce for a matching customer account , we can do so by passing in the phone number from Avaya Contact Center and Salesforce can then return to Avaya Contact Center any record we specify from the account. If we specify a return value of say customer priority (high, low or medium) we can then use this value in deciding what skillset or agent to assign the customer contact. This is the fundamental basis for performing a lookup of an external database in order to make a decision on work assignment.

For the purpose of this paper we will now step through the "coding" required on Salesforce which delivers a solution to the following requirement:

> *"Develop a Salesforce Apex Class and associated code which allows Avaya Contact Center query a Salesforce customer record for the value of a customer field containing the "Preferred Agent" and use the response value to perform a "Queue to Agent ID"*

The first part of this task is to create a new variable in Salesforce to store the value of for Agent ID in each customer record. Salesforce does not contain a placeholder for agent ID by default and therefore it must be created as new.

Within the user account, select "Customise -→ Accounts → Fields". This is shown in the following image.


Account Fields ~ salesforce.com - Developer Edition - Windows Internet Explorer
https://na7.salesforce.com/p/setup/layout/LayoutFieldList?type=Account&setupid=AccountFields&retURL=%2Fui%
File Edit View Favorites Tools Help
Favorites
Account Fields ~ salesforce.com - Developer Edition
Tom Eustace
Help
Call Center
Home Chatter Accounts Contacts Cases Solutions Reports Dashboards Documents Merchandise
Internet | Protected Mode: Off
100%

Personal Setup

- My Personal Information
- Email
- Import
- Desktop Integration
- Call Center Settings
- My Chatter Settings

App Setup

- Customize
  - Tab Names and Labels
  - Home
  - Activities
  - Campaigns
  - Leads
  - Accounts
    - Fields
    - Related Lookup Filters
    - Validation Rules
    - Triggers

Account Fields

Help for this Page

This page allows you to specify the fields that can appear on the Account page. You can create up to 500 Account custom fields.

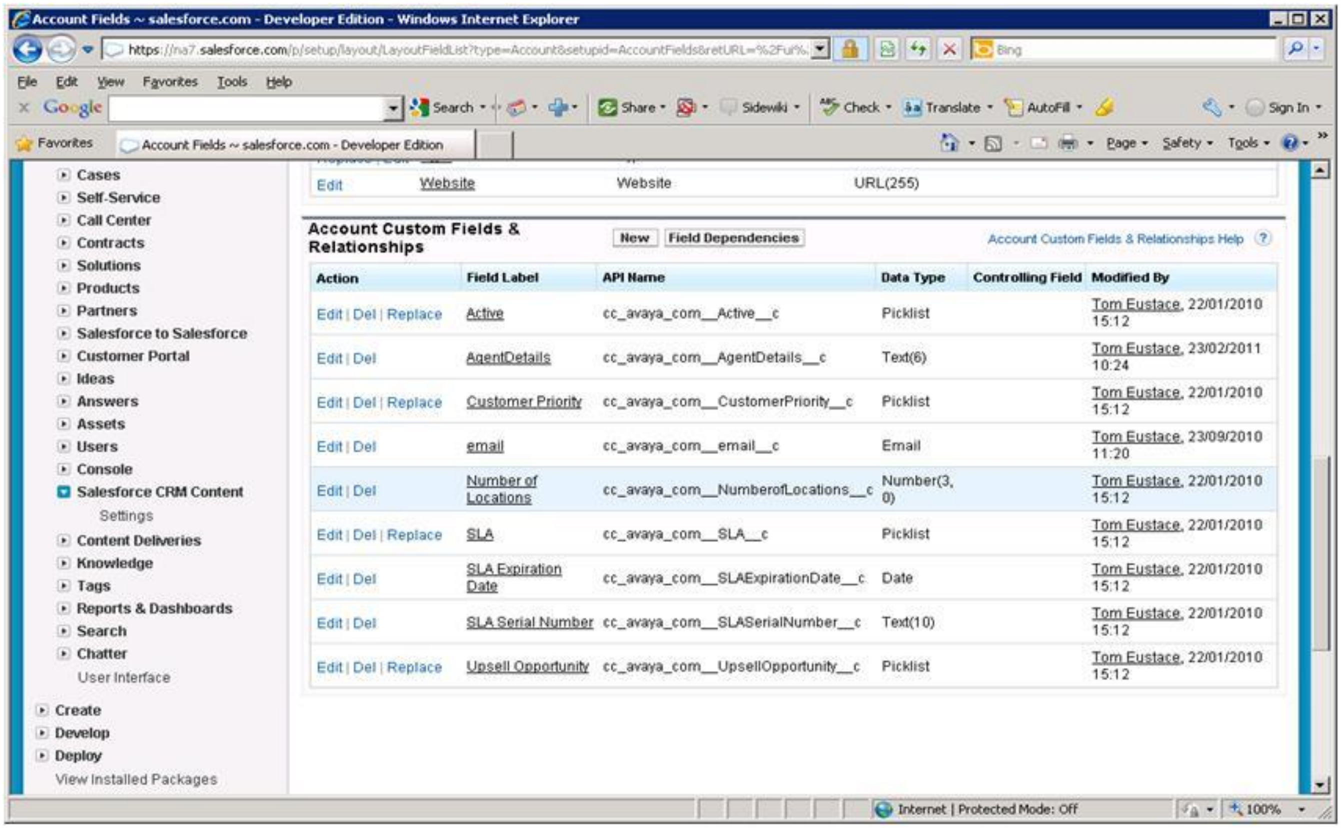
Note that deleting a custom field will delete any filters that use the custom field. It may also change the result of Assignment or Escalation Rules that rely on the custom field data.

Set History Tracking

Account Standard Fields — Account Standard Fields Help

Action	Field Label	Field Name	Data Type	Controlling Field
	Account Name	Name	Name	
Edit	Account Number	AccountNumber	Text(40)	
Edit	Account Owner	Owner	Lookup(User)	
Edit	Account Site	Site	Text(80)	
Edit	Annual Revenue	AnnualRevenue	Currency(18, 0)	
	Billing Address	BillingAddress	Address	
	Created By	CreatedBy	Lookup(User)	
Edit	Description	Description	Long Text Area(32000)	
Edit	Employees	NumberOfEmployees	Number(8, 0)	
Edit	Fax	Fax	Fax	

Scroll down to the bottom of the page to “Account Custom fields and Relationships” as shown below.


Account Fields ~ salesforce.com - Developer Edition - Windows Internet Explorer
https://na7.salesforce.com/p/setup/layout/LayoutFieldList?type=Account&setupid=AccountFields&retURL=%2Fui%...
File Edit View Favorites Tools Help
Search Share Sidewiki Check Translate AutoFill Sign In
Favorites
Account Fields ~ salesforce.com - Developer Edition
Page Safety Tools
Cases
Self-Service
Call Center
Contracts
Solutions
Products
Partners
Salesforce to Salesforce
Customer Portal
Ideas
Answers
Assets
Users
Console
Salesforce CRM Content
Settings
Content Deliveries
Knowledge
Tags
Reports & Dashboards
Search
Chatter
User Interface
Create
Develop
Deploy
View Installed Packages
New
Field Dependencies
Account Custom Fields & Relationships Help
Internet | Protected Mode: Off
100%

Edit	Website	Website	URL(255)

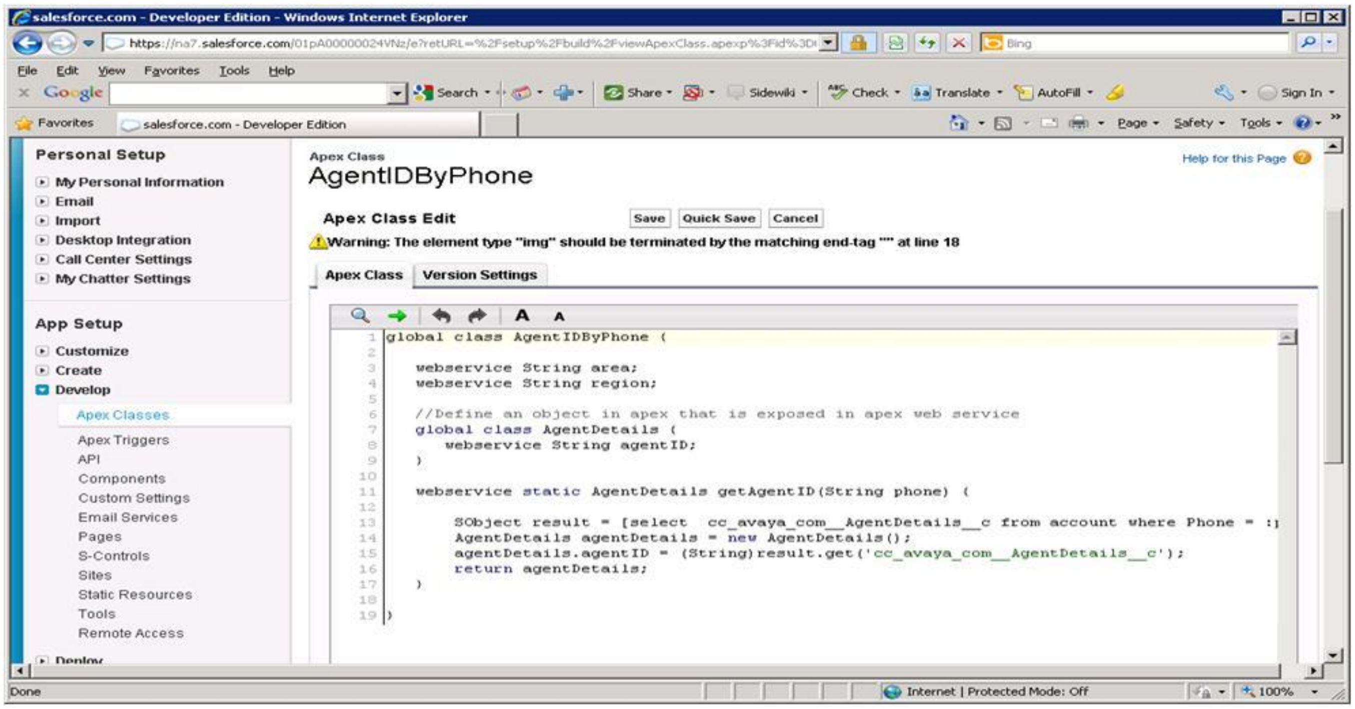
Account Custom Fields & Relationships

Action	Field Label	API Name	Data Type	Controlling Field	Modified By
Edit | Del | Replace	Active	cc_avaya_com__Active__c	Picklist		Tom Eustace, 22/01/2010 15:12
Edit | Del	AgentDetails	cc_avaya_com__AgentDetails__c	Text(6)		Tom Eustace, 23/02/2011 10:24
Edit | Del | Replace	Customer Priority	cc_avaya_com__CustomerPriority__c	Picklist		Tom Eustace, 22/01/2010 15:12
Edit | Del	email	cc_avaya_com__email__c	Email		Tom Eustace, 23/09/2010 11:20
Edit | Del	Number of Locations	cc_avaya_com__NumberofLocations__c	Number(3, 0)		Tom Eustace, 22/01/2010 15:12
Edit | Del | Replace	SLA	cc_avaya_com__SLA__c	Picklist		Tom Eustace, 22/01/2010 15:12
Edit | Del	SLA Expiration Date	cc_avaya_com__SLAExpirationDate__c	Date		Tom Eustace, 22/01/2010 15:12
Edit | Del	SLA Serial Number	cc_avaya_com__SLASerialNumber__c	Text(10)		Tom Eustace, 22/01/2010 15:12
Edit | Del | Replace	Upsell Opportunity	cc_avaya_com__UpsellOpportunity__c	Picklist		Tom Eustace, 22/01/2010 15:12

In the example above, Avaya has created and defined a placeholder called "AgentDetails" which is a new field in each customer record. In it we will store the agent ID of the last agent associated with the customer account. If you wish to lookup Salesforce for customer email fields, you will have to add an field to store customer's email address.

Now we need to develop a new code module which will take a value (customer telephone number from Avaya Contact Center) and then perform a lookup of that value in Salesforce. Upon a successful match, the code module will return a value which will be the agent ID (if present) which last handled the customer call.

In you Salesforce account on the left hand side, click on Develop → Apex Classes. In the Avaya account there is a listing of all of the code modules used by Avaya. One of them is called "AgentIDByPhone".



Apex Class
AgentIDByPhone

Apex Class Edit

Warning: The element type "img" should be terminated by the matching end-tag "" at line 18

```
global class AgentIDByPhone {

   webservice String area;
   webservice String region;

   //Define an object in apex that is exposed in apex web service
   global class AgentDetails {
      webservice String agentID;
   }

   webservice static AgentDetails getAgentID(String phone) {

       SObject result = [select  cc_avaya_com__AgentDetails__c from account where Phone = :p
       AgentDetails agentDetails = new AgentDetails();
       agentDetails.agentID = (String)result.get('cc_avaya_com__AgentDetails__c');
       return agentDetails;
   }

}
```

The above image is taken directly from within the Avaya account in the Apex programming interface in Salesforce. The code module (AgentIDByPhone) was created by Avaya in the Avaya account on Salesforce. It is only approx 10 lines of code.

The important line is as follows:

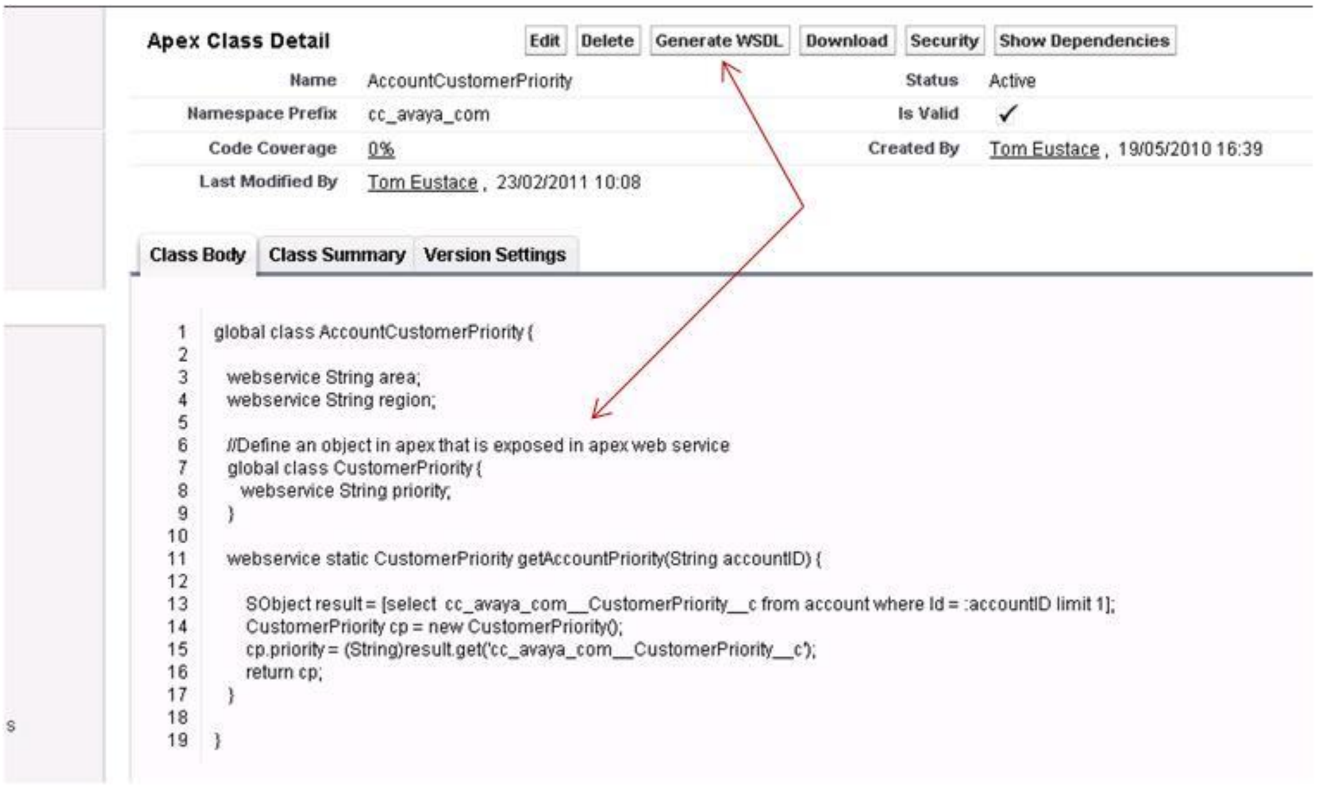
SObject result = (select cc_avaya_com__AgentDetails__c from account where Phone = :)

In simple terms what this line of code is doing is performing a lookup of the customer account and searching for a phone number value passed in from Avaya Contact Center. If a match is found, the query will return the value associated with the Agent Details field (previously created). This value is simply the Preferred Agent that handled this customer with this telephone number.

The remaining lines of code set the agent ID value into a variable and return it back to Avaya Contact Center. Avaya Contact Center then takes this value and performs a work assignment decision with it.

Within the Avaya developer account, there are other examples of code which perform various functions. One of these modules takes in a customer’s phone number from Avaya Contact Center, performs a look in Salesforce for a matching customer phone number and if successful, returns a priority for that customer to Avaya Contact

Center. This priority is then used to make an assignment decision on which skillset to assign the call i.e. high priority or low priority. Here is the actual code.



Name	AccountCustomerPriority	Status	Active
Namespace Prefix	cc_avaya_com	Is Valid	✓
Code Coverage	0%	Created By	Tom Eustace , 19/05/2010 16:39
Last Modified By	Tom Eustace , 23/02/2011 10:08		

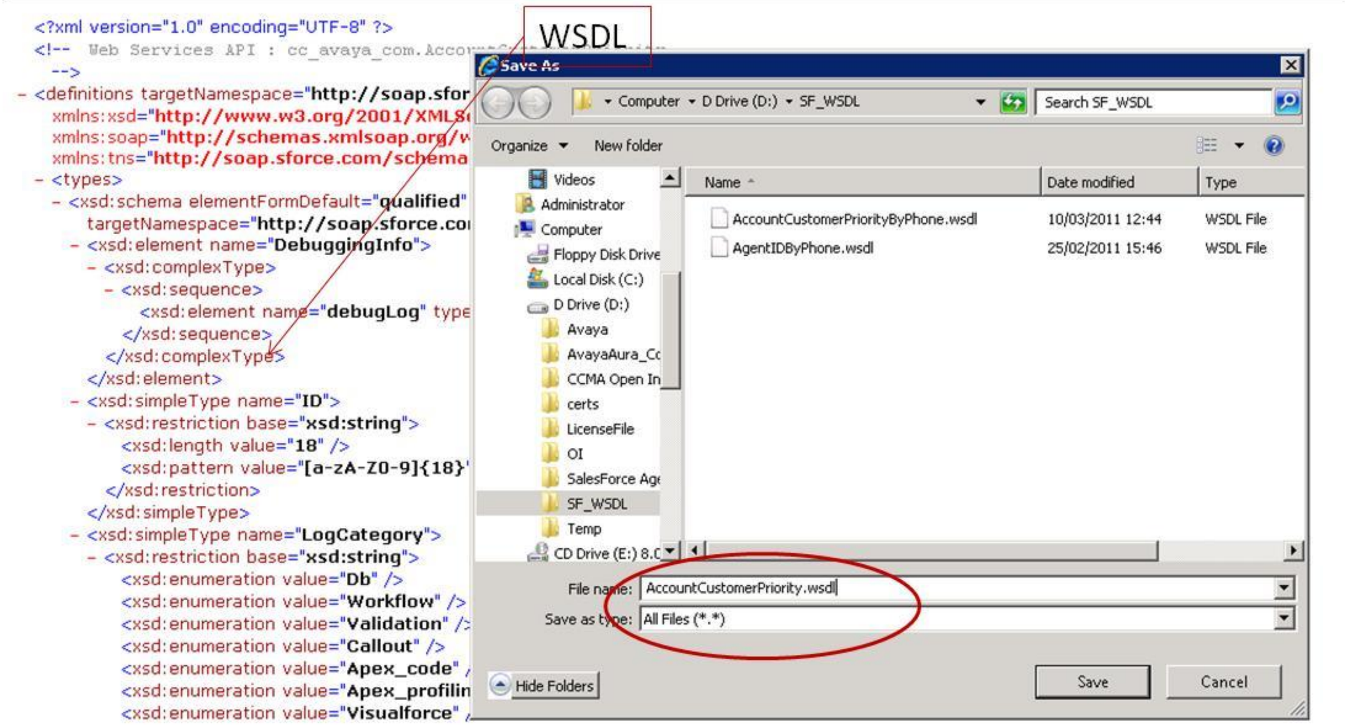
```
 1  global class AccountCustomerPriority {
 2
 3    webservice String area;
 4    webservice String region;
 5
 6    //Define an object in apex that is exposed in apex web service
 7    global class CustomerPriority {
 8      webservice String priority;
 9    }
10
11    webservice static CustomerPriority getAccountPriority(String accountID) {
12
13       SObject result = [select  cc_avaya_com__CustomerPriority__c from account where Id = :accountID limit 1];
14       CustomerPriority cp = new CustomerPriority();
15       cp.priority = (String)result.get('cc_avaya_com__CustomerPriority__c');
16       return cp;
17    }
18
19  }
```

Lines 6, 7 and 8 define the field "CustomerPriority" as an object that is exposed in a web service. External applications can call this web service which then returns the value of this object from Salesforce in a matched customer account.

Line 13 performs a lookup of Salesforce accounts for a telephone number matching the telephone number passed in from Avaya Contact Center. If a successful match is found, the value of the customer priority filed is returned in the web service.

The code module above and the previous code module can now be exported into a form that Avaya Contact Center can utilise within its workflow. That form is termed "WSDL" or Web Services Descriptor Language. If the code module compiles correctly in Salesforce, then the developer can export it as a WSDL and import that WSDL into Avaya Contact Center. It is assumed that the reader of this document is familiar with these concept s and terms.

In order to invoke the above code module using a web service in Avaya Contact Center, we need to generate the corresponding WSDL file. On the top of the development screen, the user clicks on “Generate WSDL”. This produces the WSDL output which can then be downloaded from Salesforce.



```
<?xml version="1.0" encoding="UTF-8" ?>
<!--  Web Services API : cc_avaya_com.Accou
-->
- <definitions targetNamespace="http://soap.sfor
   xmlns:xsd="http://www.w3.org/2001/XMLS
   xmlns:soap="http://schemas.xmlsoap.org/w
   xmlns:tns="http://soap.sforce.com/schema
 - <types>
  - <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://soap.sforce.co
   - <xsd:element name="DebuggingInfo">
    - <xsd:complexType>
     - <xsd:sequence>
         <xsd:element name="debugLog" type
       </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   - <xsd:simpleType name="ID">
    - <xsd:restriction base="xsd:string">
        <xsd:length value="18" />
        <xsd:pattern value="[a-zA-Z0-9]{18}'
      </xsd:restriction>
    </xsd:simpleType>
   - <xsd:simpleType name="LogCategory">
    - <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Db" />
        <xsd:enumeration value="Workflow" />
        <xsd:enumeration value="Validation" />
        <xsd:enumeration value="Callout" />
        <xsd:enumeration value="Apex_code" /
        <xsd:enumeration value="Apex_profilin
        <xsd:enumeration value="Visualforce" /
```

Name	Date modified	Type
AccountCustomerPriorityByPhone.wsdl	10/03/2011 12:44	WSDL File
AgentIDByPhone.wsdl	25/02/2011 15:46	WSDL File

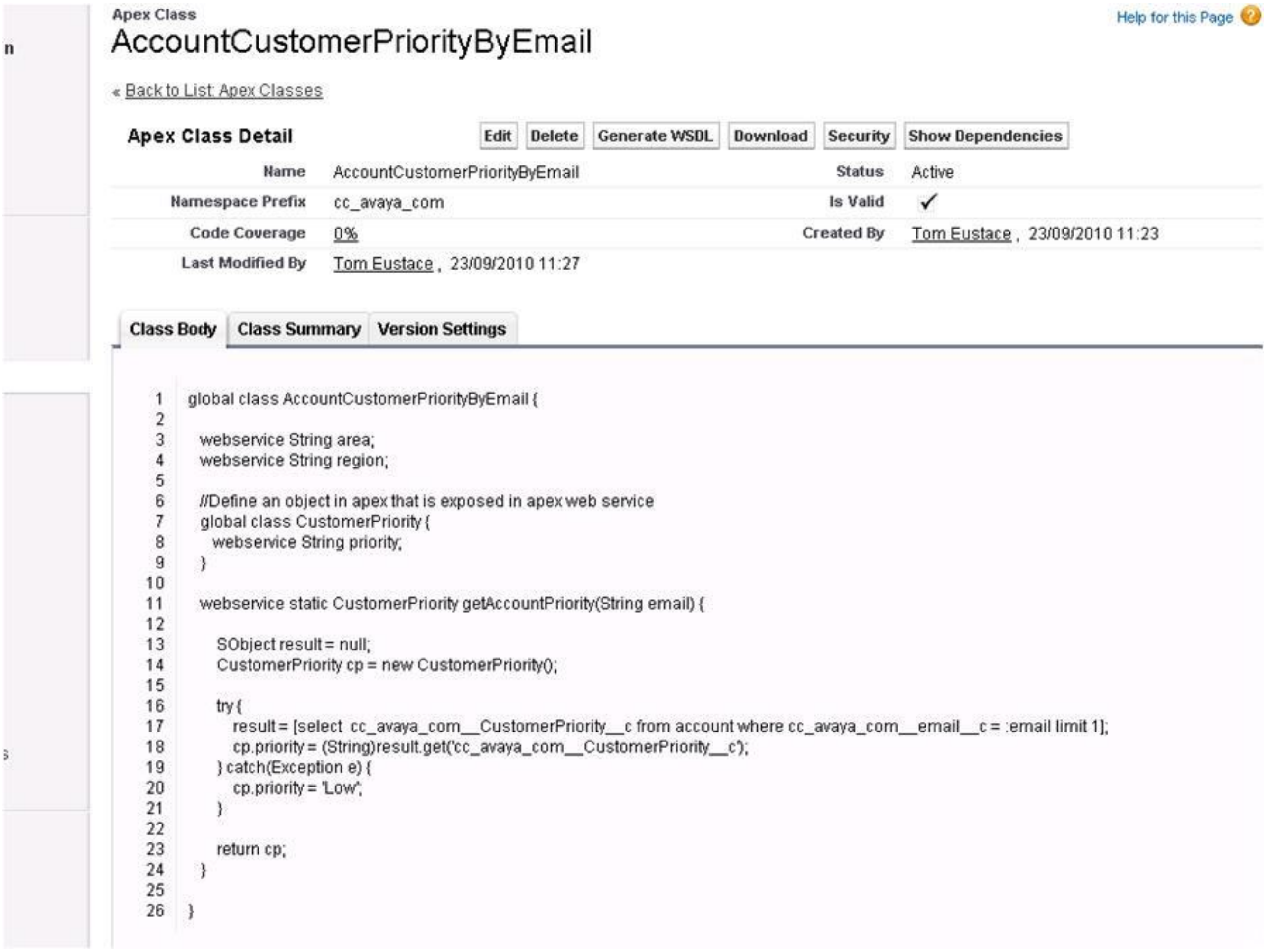
File name: AccountCustomerPriority.wsdl

Save as type: All Files (*.*)

The above images show the sample WSDL for the code module in Salesforce which is displayed in a browser window. Using the browser controls, select “File → Save as” and in the Save As pop up, select “All Files (*.*) and change filename to *.wsdl. Save the WSDL file to a folder on your Avaya Contact Center server.

We will import this WSDL into Avaya Contact Center for use in the Orchestration Designer application and this will be described in the next section. All of the code modules we have created in Salesforce can be exposed as a web service and downloaded as a WSDL using the above methodology.

Examples of some other Apex Classes are now shown in the following screen shots – sample code given in Appendix 4



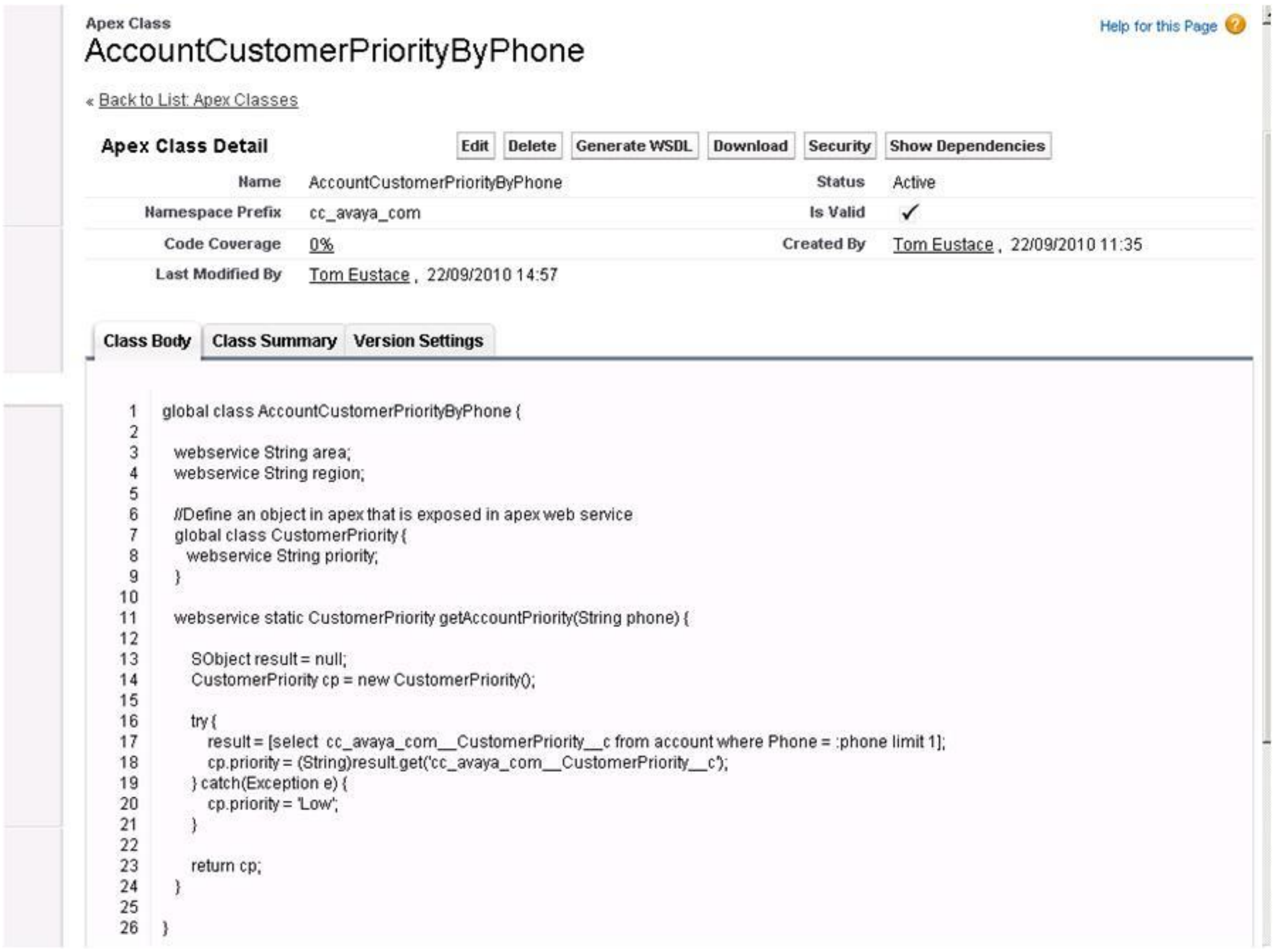
AccountCustomerPriorityByEmail

« Back to List: Apex Classes

Apex Class Detail

Name	AccountCustomerPriorityByEmail	Status	Active
Namespace Prefix	cc_avaya_com	Is Valid	✓
Code Coverage	0%	Created By	Tom Eustace , 23/09/2010 11:23
Last Modified By	Tom Eustace , 23/09/2010 11:27		

```
global class AccountCustomerPriorityByEmail {

  webservice String area;
  webservice String region;

  //Define an object in apex that is exposed in apex web service
  global class CustomerPriority {
    webservice String priority;
  }

  webservice static CustomerPriority getAccountPriority(String email) {

     SObject result = null;
     CustomerPriority cp = new CustomerPriority();

     try {
        result = [select cc_avaya_com__CustomerPriority__c from account where cc_avaya_com__email__c = :email limit 1];
        cp.priority = (String)result.get('cc_avaya_com__CustomerPriority__c');
     } catch(Exception e) {
        cp.priority = 'Low';
     }

     return cp;
  }

}
```



Apex Class
AccountCustomerPriorityByPhone

« Back to List: Apex Classes

Apex Class Detail

Name	AccountCustomerPriorityByPhone	Status	Active
Namespace Prefix	cc_avaya_com	Is Valid	✓
Code Coverage	0%	Created By	Tom Eustace , 22/09/2010 11:35
Last Modified By	Tom Eustace , 22/09/2010 14:57		

```
global class AccountCustomerPriorityByPhone {

  webservice String area;
  webservice String region;

  //Define an object in apex that is exposed in apex web service
  global class CustomerPriority {
    webservice String priority;
  }

  webservice static CustomerPriority getAccountPriority(String phone) {

     SObject result = null;
     CustomerPriority cp = new CustomerPriority();

     try {
        result = [select cc_avaya_com__CustomerPriority__c from account where Phone = :phone limit 1];
        cp.priority = (String)result.get('cc_avaya_com__CustomerPriority__c');
     } catch(Exception e) {
        cp.priority = 'Low';
     }

     return cp;
  }

}
```

6 – Avaya Contact Center Database Integration Wizard (DIW)

In order to exchange data to/from Avaya Contact Center to Salesforce using web services, we need to configure the web service connection between the two applications. In a previous section, we described how to configure the logon creditentials to Salesforce from Avaya Contact Center. This section describes how the web service is invoked and how it is configured on Avaya Contact Center to be used within the Orchestration Designer workflows.

Avaya Contact Center contains a pre-packaged application called Database Integration Wizard (DIW) for exchanging information with internal/external database systems. DIW also allows customers to import third Party Web Services into Avaya Contact Center where they can be accessed using a host block inside the Orchestration Designer Environment. This functionality has been extended to support Salesforce. When a customer has created an Apex Web Service in Salesforce, customers can import the associated WSDL using DIW.

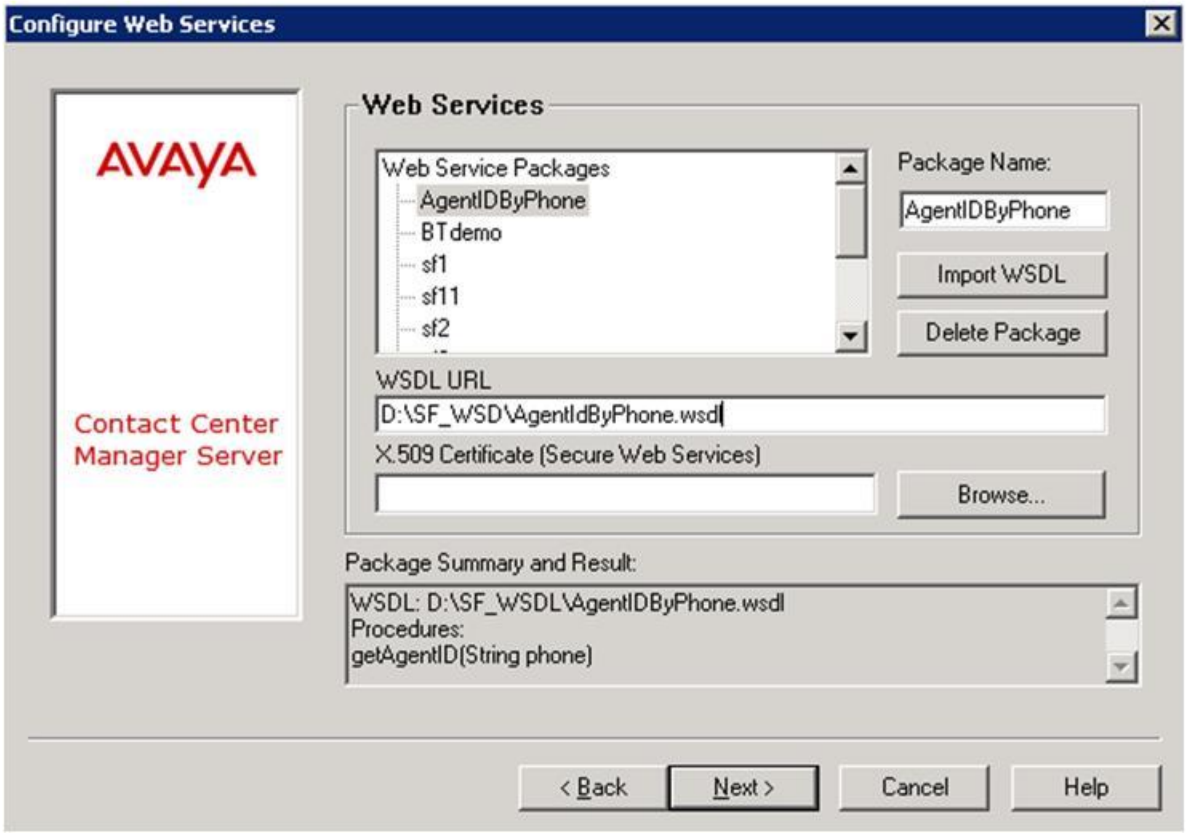
We will describe now how to use DIW to import the web service we want to use for communicating with Salesforce. We will use the web service we created in the last section to query Salesforce for the "last agentID" based on customer's phone number.

On the CCMS, go to Start/Programs/Avaya/Contact Center/Manager Server/Database Integration Wizard.

The Introduction form is displayed which read "Welcome to the CCMS Database Integration Wizard"

Click Next to go to the next screen. In the screen below you will be required to enter a "Provider ID". This provider ID will be used by Orchestration Designer to invoke the correct web service. Avaya recommends a provider ID value between <1 to 10>. In our example we have chosen a provider ID of 5.

Now click "Next" until the window titled "Configure Web Services" appears. In this screen, you will point to the WSDL file you downloaded in the previous section. You will be required to (i) enter a package name of your choosing which is a textual description of the web service you plan to import (ii) Navigate to the web service location which can be a URL or a local file as shown in our example. Note the "browse" button applies to the certificate for secure web services. In our example we are not using secure web services instead we are using a local file and the path to this file must be typed in manually.



Package Name: AgentIDByPhone

WSDL URL: D:\SF_WSD\AgentIdByPhone.wsdl

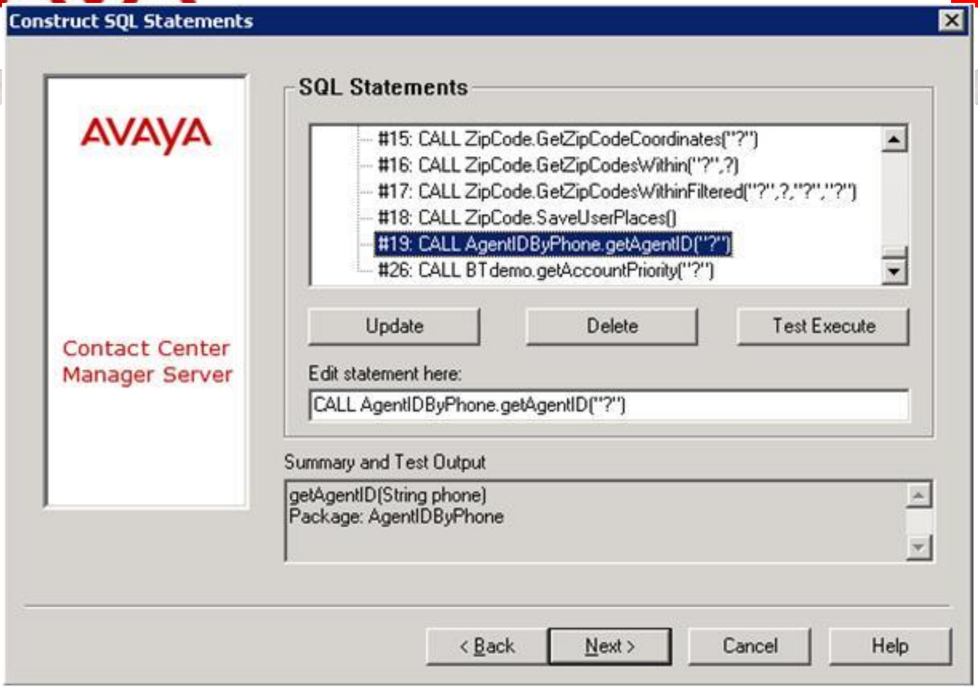
X.509 Certificate (Secure Web Services):

Package Summary and Result:

```
WSDL: D:\SF_WSDL\AgentIDByPhone.wsdl
Procedures:
getAgentID(String phone)
```

In the location for "WSDL URL", type the path to the location of the WSDL file. Then click on "Import WSDL". If the WSDL is located and is correct, then a successful import will be displayed in the bottom result pane.

In the above example, the import was successful as the web service which we created earlier is clearly displayed. Click Next to move to the "Construct SQL Statements". Scroll down to the bottom of this section to the web services. If the previous import was successful, you will see a listing for your web service.

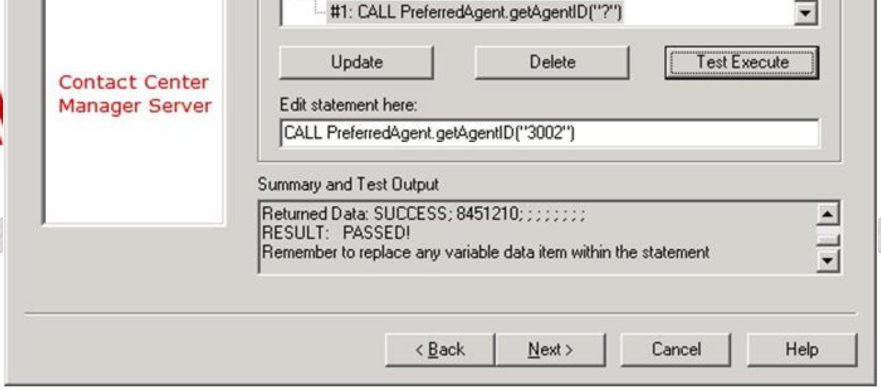


```
#15: CALL ZipCode.GetZipCodeCoordinates("?")
#16: CALL ZipCode.GetZipCodesWithin("?",?)
#17: CALL ZipCode.GetZipCodesWithinFiltered("?",?,"?","?")
#18: CALL ZipCode.SaveUserPlaces()
#19: CALL AgentIDByPhone.getAgentID("?")
#26: CALL BTdemo.getAccountPriority("?")
```

Edit statement here: CALL AgentIDByPhone.getAgentID("?")

```
getAgentID(String phone)
Package: AgentIDByPhone
```

The previous screen grab shows our new web service “AgentIdByPhone”. Also displayed is the actual command or statement executed by the web service. We can test this statement by replacing the “?” with an actual value in this case a phone number. For example, if we replace the “?” with a number i.e. 3002 and then click on ***Test Execute***, this action will execute the statement using 3002 directly on Salesforce and return the AgentID value from the customer account with telephone number 3002 from Salesforce. See following example.



CALL PreferredAgent.getAgentID("3002")

```
Returned Data: SUCCESS; 8451210; ; ; ; ; ; ; ;
RESULT:   PASSED!
Remember to replace any variable data item within the statement
```

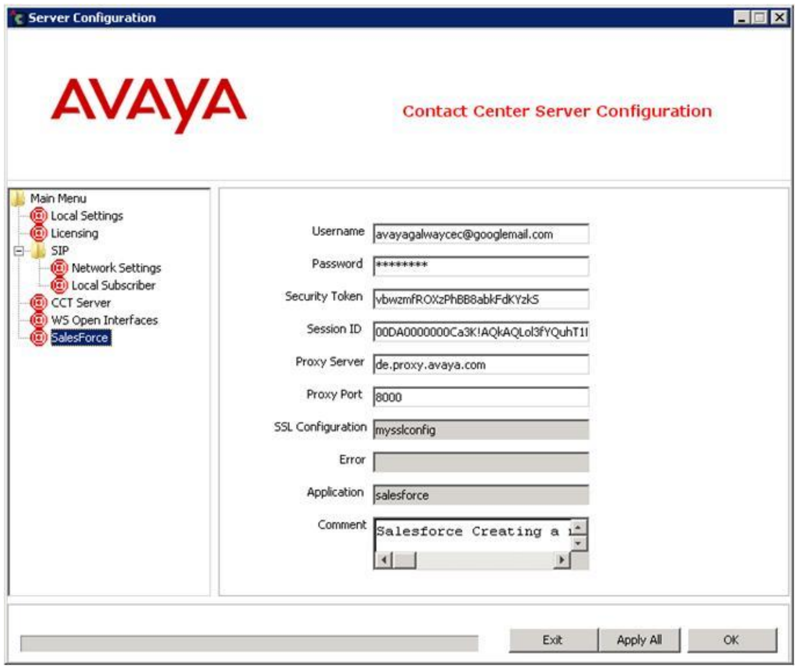
The above screen shot shows a successful test in that the preferred agent id 8451210 is returned for a calling number of 3002 and also the RESULT field is shown as SUCCESS.

With this test, we can evaluate and prove our connection to Salesforce before we put it into an Orchestration Designer workflow. We will now also check that the CCMS server configuration is correctly populated with the new access tokens from Salesforce.

Note you will require the integer value to the left of the web service (in the above example (#1) when you are configuring the Host data block in Orchestration Designer in the next section.

Click Next and the Finish to close and exit out of the DIW application. Once this step is completed, we now can move to the next section where we will define the Orchestration Designer workflow which enables all of this connectivity.

Now we need to verify our CCMS server configuration is working as programmed from Chapter 4. On the Avaya Contact Center server, go to Start/Programs/Avaya/Contact Center Manager Server/Server Configuration. Once open, click on the "Salesforce" plug-in tab. You should now see an entry for the SESSION ID. You will only see this when the web service call into Salesforce returns a valid parameter. For example if there is no match of calling number = 3002, then this dialog box in the CCMS server configuration will not be populated.



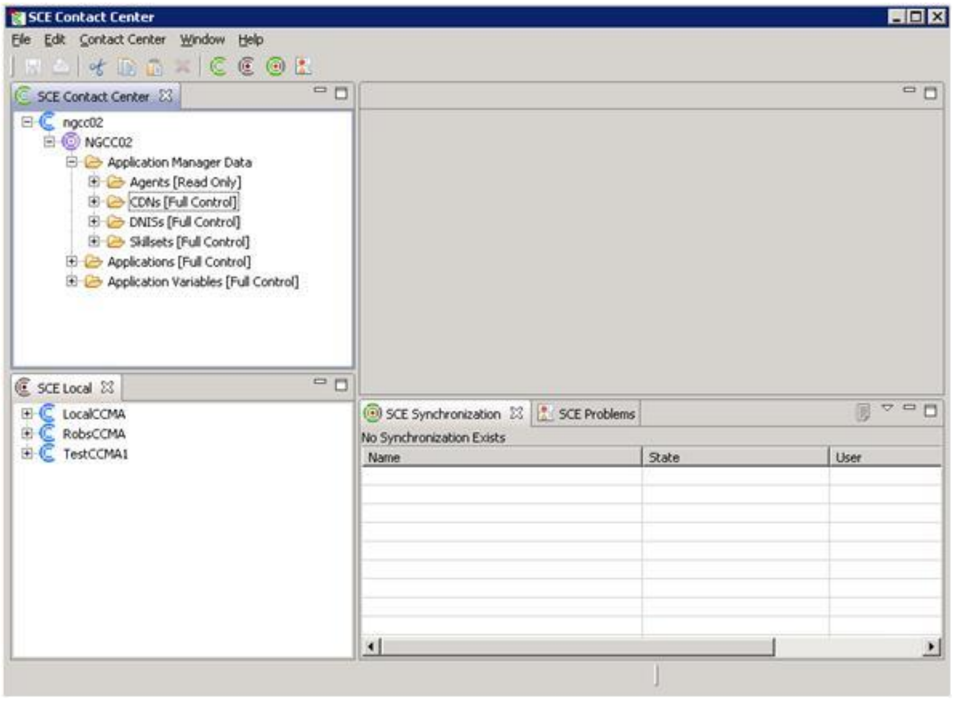
Username	avayagalwaycec@googlemail.com
Password	********
Security Token	vbwzmfROXzPhBB8abkFdKYzkS
Session ID	00DA0000000Ca3K!AQkAQLol3fYQuhT1I
Proxy Server	de.proxy.avaya.com
Proxy Port	8000
SSL Configuration	mysslconfig
Error	
Application	salesforce
Comment	Salesforce Creating a

On Server Configuration, Click Exit and then YES to close this application. DO not select OK or Apply ALL.

7 – Defining Orchestration Designer Workflow

Up until this point, we have looked at creating the connections between Avaya Contact Center and Salesforce using a number of different interfaces. We will now define the Orchestration Designer workflow which can use these interfaces to exchange information. It is presumed that the reader is familiar with Orchestration Designer (Service Creation Environment) and knows how to use the application to define appropriate workflows.

To access Orchestration, login to CCMA and within the Scripting option, open the Orchestration Designer application builder as show below.



Our new Orchestration Designer workflow must first pass across to the Salesforce application the customer phone number or calling number (A-Number value – again this could be an email address etc.). Salesforce will then perform a lookup of its customer records for the customer phone number and respond with an appropriate value (in our example the Preferred Agent) which we will then use in our flow to make a work assignment decision. There are a number of steps to building this workflow.

(a) Document what we want it to do....the flow logic etc.

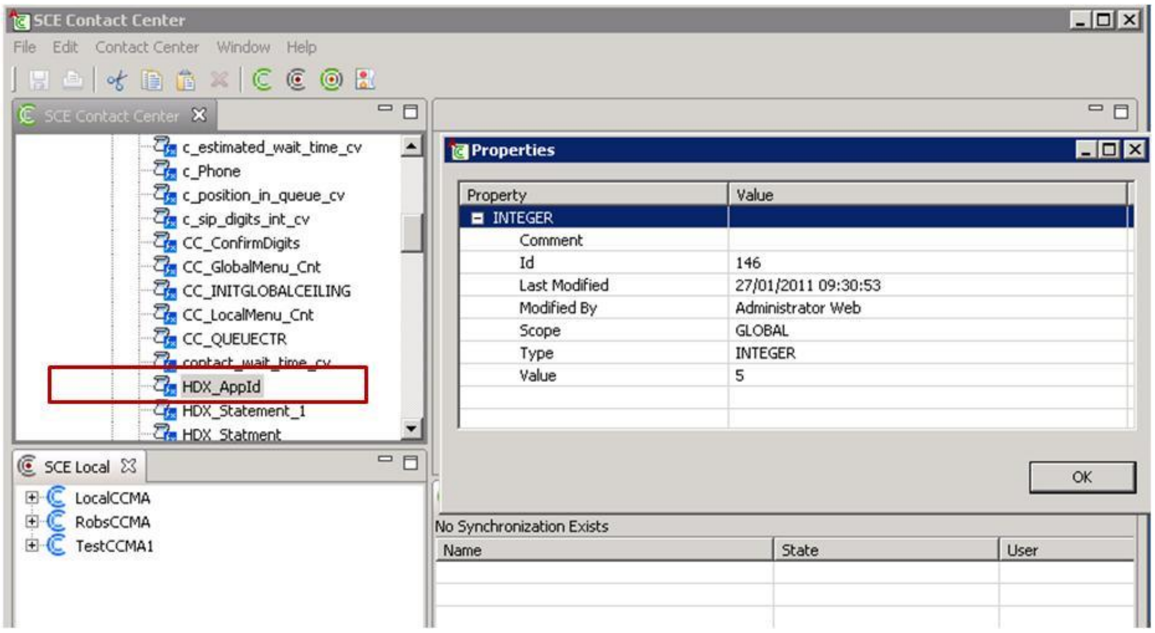
(b) Create the appropriate contact variables in Avaya Contact Center, which will be used to store information being passed to Salesforce and back from Salesforce.
(c) Active our new flow once completed.

Defining Contact Variables

In our new workflow we will require 4 variables plus one intrinsic in our Orchestration Designer workflow.

(i) Integer Variable 1 to specify the DIW Provider ID value we created earlier in DIW. We used "HDX_AppID.
(ii) Integer Variable 2 to specify which Web Service we wish to use in our request to Salesforce. Each Web service is given a unique value when it is imported into Avaya Contact Center using DIW. We used "Web_Service_Req_cv" as the variable name to store the value. In our example, the web service we require is "1" – see previous section.
(iii) Avaya Contact Center Intrinsic to select the customer's phone number or any data value we are sending to Salesforce via DIW. We used the system intrinsic "CLID" (calling number).
(iv) String Variable 4 to store the response back from Salesforce indicating success or fail. We used "HDX_Resp_cv"
(v) AGENT_ID type Variable 5 to store the response value back from Salesforce. We used "contact_agent_cv".

The first variable will be used to specify an Integer value of the DIW provider ID for Salesforce connector. The following image is an example taken from the Avaya Contact Center used for this integration.

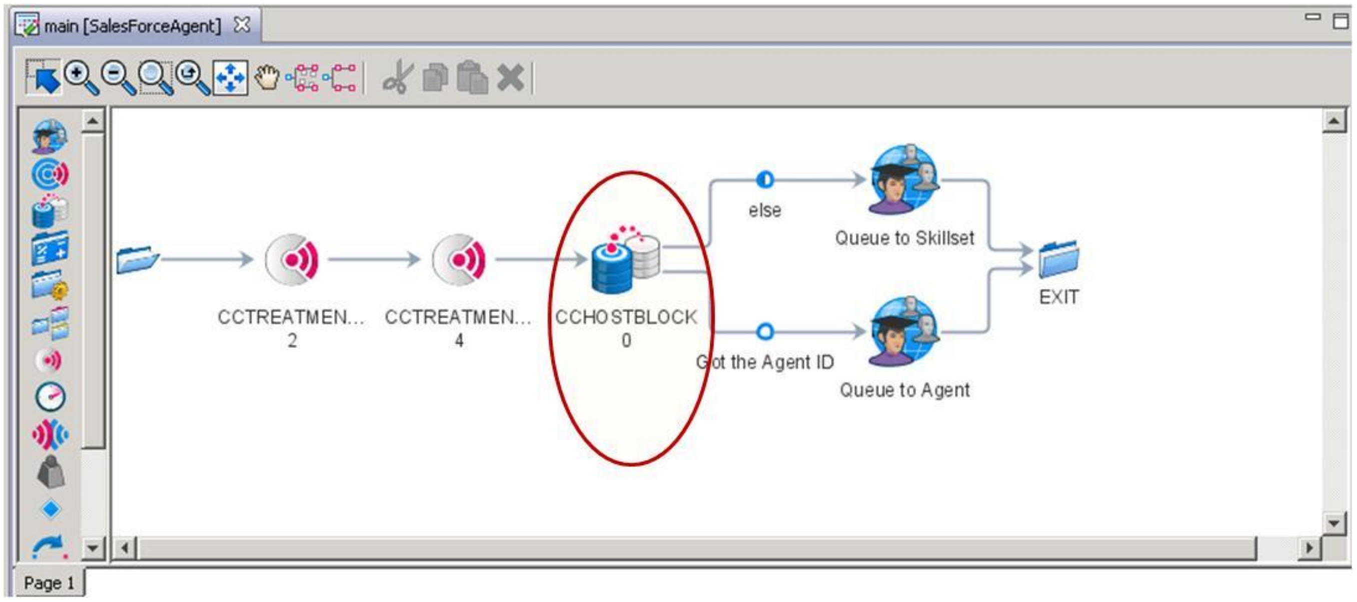


Property	Value
INTEGER	
Comment	
Id	146
Last Modified	27/01/2011 09:30:53
Modified By	Administrator Web
Scope	GLOBAL
Type	INTEGER
Value	5

It is subjective as to which order you create the variables and the workflow. You can create the variables as you build the workflow or before you build the workflow. The workflow we used for this paper is shown below.

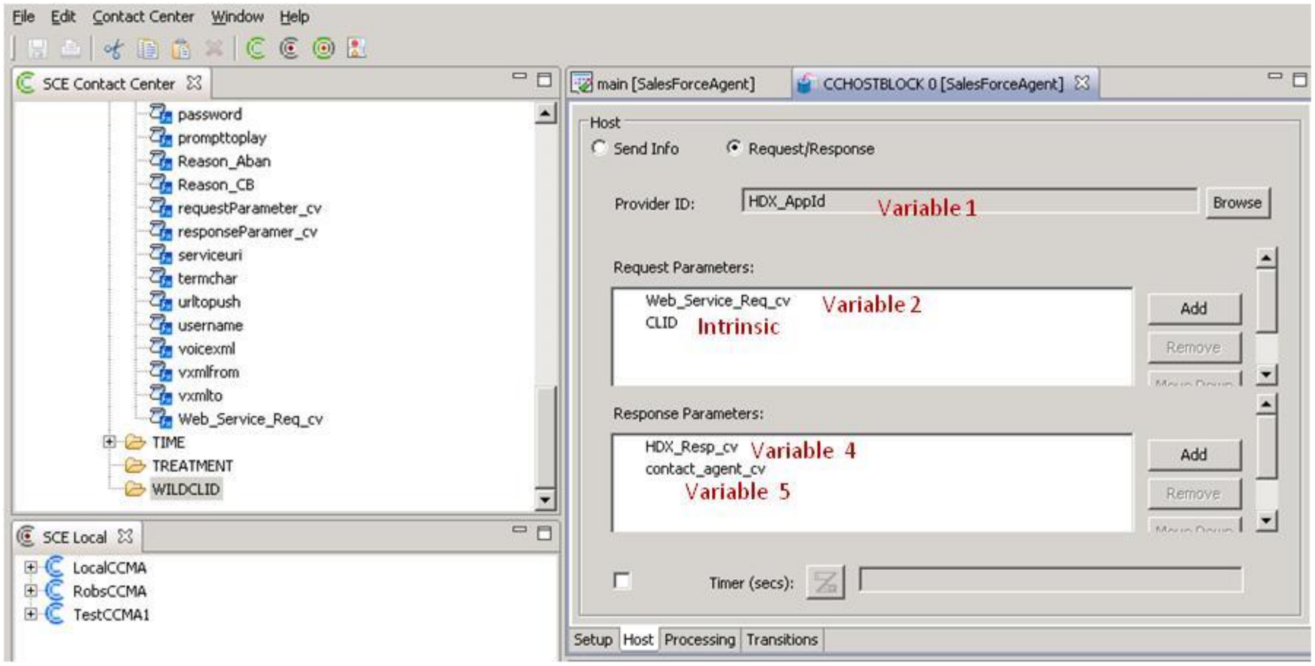
Programming Orchestration Designer workflow and HOST Block

Below is the Orchestration Designer workflow that was used to create the “preferred agent” workflow. All voice contacts entering this workflow will first be given a treatment (music/announcement). Then before we select the agent, we will pass across to Salesforce the caller’s phone number. Salesforce will respond with a value representing the last agent who handled a call from this number. The workflow checks a successful match in Salesforce. If the returned result is “success” then we perform a “Queue to Agent” with the agent ID parameter returned from Salesforce. If there is a “fail” response from Salesforce, then the Orchestration Designer flow simply assigns the caller to the skillset for normal processing.



The block which performs the lookup in Salesforce for the agent ID is the *host* block. Once the *host* block has been added to a *flow*, it needs to be configured to point to the appropriate web service that was imported using DIW. In our example the imported web service ID is 19 – see previous section.

Outlined above in red is the Host block. To program the host block, right click in it, select OPEN and program as follows:



Send Info [✓ Request/Response]

Provider ID: HDX_AppId Variable 1

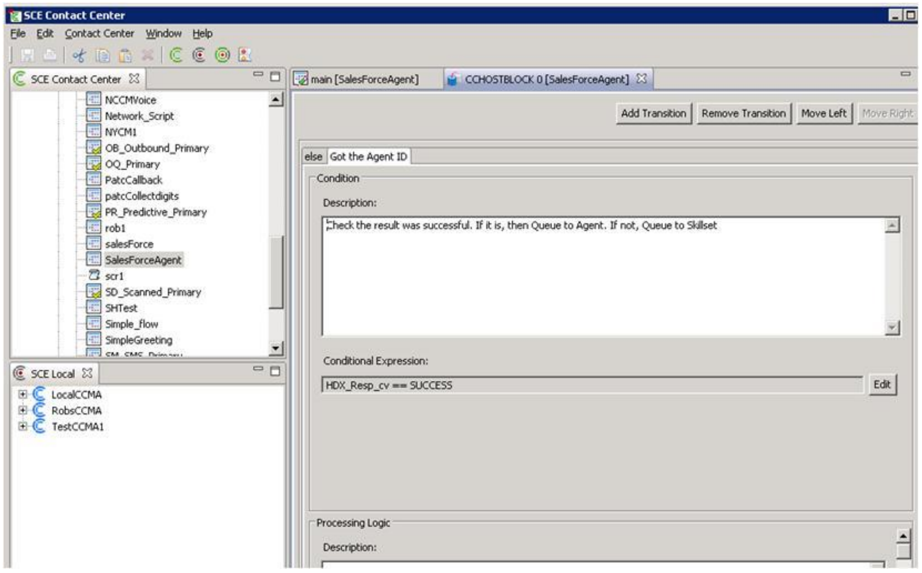
Request Parameters:
Web_Service_Req_cv Variable 2
CLID Intrinsic

Response Parameters:
HDX_Resp_cv Variable 4
contact_agent_cv
Variable 5

Timer (secs):

In the above example, we show where each of the variables and intrinsic are programmed in our host block on the Avaya Contact Center solution.

The other key functionality of the Host Block is to check if there is a successful response to the Salesforce query. If there is, then the message “Success” will be returned in Variable 4. We then check for this response and if found, we perform a “Queue to Agent”. If not then we perform a standard skillset assignment operation.



Description: Check the result was successful. If it is, then Queue to Agent. If not, Queue to Skillset

Conditional Expression: HDX_Resp_cv == SUCCESS

In the above example the HDX-Resp_cv==Success means we have a successful match of the customers phone number in Salesforce and this response is returned to Avaya Contact Center. This is one leg of the transition from the host block to the next block. If no match is found, then we follow the “else” path to the next Orchestration Designer block.

If there are errors in your Orchestration Designer flow, you will see them on the bottom of Orchestration Designer under the heading “Orchestration Designer problems”. If there are no errors, then the workflow will save successfully and you can then activate using the normal Orchestration Designer procedures.

In Part I of this white paper we showed how to screen pop the actual customer details from Salesforce when the agent answers the voice contact. The same procedures can be used to generate a screen pop of the customer record from Salesforce. As this detail is already described we now can refer readers to Part I of this document.

8 – SUMMARY

This short application note provides a set of instructions for developing integration between Avaya Contact Center and Salesforce using open standards programming and web services.

The workflow described allows Avaya Contact Center to lookup in Salesforce the agent which last handled a customer contact based on the customer phone number. If a successful match is found, then that agent (if available) will be assigned the customer voice contact as a first choice. If not, then the voice contact is assigned to the skillset group.

Additional information on Salesforce and Avaya Contact Center integration can be downloaded from support at Avaya.com

APPENDIX 1 – LOOKUPBYPARM SOURCE CODE

The LookupByUrlParam piece is a custom piece of code that Avaya have made available to any Salesforce developer. It passes in a key and a value. The key in this instance is “phone”.

Within Salesforce, under App Setup > Develop > Pages, we created the following Visual Force Page (LookupByUrlParam):

```
<apex:page controller="LookupByUrlParamController" action="{!redirectToAccount}">
  Retrieving data and redirecting...
</apex:page>
```

This calls an Apex Class that was created under App Setup > Apex Classes. See following lines of actual code we used. (Can be saved and opened in any text editor and used)

```
public class LookupByUrlParamController {

    String accountName;
    String accountNumber;
    String phone;
    String website;
    String email;

    public LookupByUrlParamController () { }

    public String redirectToAccount() {

        Account account;

        Map<String,String> params = ApexPages.currentPage().getParameters();
        if(params.size() > 0) {
```

```
      accountName = params.get('account_name');
      accountNumber = params.get('account_number');
      phone = params.get('phone');
      website =  params.get('website');
      email =  params.get('email');
  }

  try {
      if(accountName != null) {
          account = [select ID from Account where name = :accountName limit 1];
      }
  } catch (System.queryException e) {//no entry found for lookup item, display empty account page
      return 'https://na7.salesforce.com/001/e';
  }

  try {
      if(accountNumber != null) {
          account = [select ID from Account where AccountNumber = :accountNumber limit 1];
      }
   } catch (System.queryException e) {//no entry found for lookup item, display empty account page
      return 'https://na7.salesforce.com/001/e';
   }

  try {
       if(phone != null) {

          String npa;
          String nnx;
          String extension;

          //  Added logic for NA phone numbers

          if (phone.length() == 10) {
              npa = phone.substring(0,3);
              nnx = phone.substring(3,6);
              extension = phone.substring(6,10);
              phone = '(' + npa + ') ' + nnx + '-' + extension;
          }

          account = [select ID from Account where phone = :phone limit 1];
       }
  } catch (System.queryException e) {//no entry found for lookup item, display empty account page
      return 'https://na7.salesforce.com/001/e';
  }
```

```
    try {
        if(website != null) {
           account = [select ID from Account where website = :website limit 1];
        }
     } catch (System.queryException e) {//no entry found for lookup item, display empty account page
        return 'https://na7.salesforce.com/001/e';
     }

     try {
         if(email != null) {
           account = [select ID from Account where email__c = :email limit 1];
         }
     } catch (System.queryException e) {//no entry found for lookup item, display empty account page
         return 'https://na7.salesforce.com/001/e';
     }



    String accountUrl;
    if(account != null) {
       accountUrl = '/' + account.Id;
    } else {
       accountUrl = '/';
    }

    return accountUrl;
}

public static testMethod void testLookupByUrlParamAccount() {
    LookupByUrlParamController controller = new LookupByUrlParamController();
    controller.accountName = 'Avaya';
    String redirectUrl = controller.redirectToAccount();
    System.assertEquals(redirectUrl, '/001A0000007UkkFIAS');
}

public static testMethod void testLookupByUrlParamInvalidAccount() {
    LookupByUrlParamController controller = new LookupByUrlParamController();
    controller.accountName = '';
    String redirectUrl = controller.redirectToAccount();
    System.assertEquals(redirectUrl, 'https://na7.salesforce.com/001/e');
}

public static testMethod void testLookupByUrlParamPhone() {
        LookupByUrlParamController controller = new LookupByUrlParamController();
        controller.phone = '1234';
        String redirectUrl = controller.redirectToAccount();
        System.assertEquals(redirectUrl, '/001A0000007UkkFIAS');
    }

     public static testMethod void testLookupByUrlParamWherePhoneNumberIs10Chars() {
        LookupByUrlParamController controller = new LookupByUrlParamController();
        controller.phone = '1234567891';
        String redirectUrl = controller.redirectToAccount();
        System.assertEquals(redirectUrl, 'https://na7.salesforce.com/001/e');//no record found
    }

    public static testMethod void testLookupByUrlParamInvalidPhoneNumber() {
        LookupByUrlParamController controller = new LookupByUrlParamController();
        controller.phone = '';
        String redirectUrl = controller.redirectToAccount();
        System.assertEquals(redirectUrl, 'https://na7.salesforce.com/001/e');//no record found
    }

    public static testMethod void testLookupByUrlParamAccountNumber() {
        LookupByUrlParamController controller = new LookupByUrlParamController();
        controller.accountNumber = '4321';
        String redirectUrl = controller.redirectToAccount();
        System.assertEquals(redirectUrl, '/001A0000007UkkFIAS');
    }

    public static testMethod void testLookupByUrlParam() {
        LookupByUrlParamController controller = new LookupByUrlParamController();
        String redirectUrl = controller.redirectToAccount();
        System.assertEquals(redirectUrl, '/');
    }

}
```

APPENDIX 2 – LOOKUPBYPARMNEW SOURCE CODE

```
public class LookupByUrlParamControllerNew {

    String customerName;
    String customerNumber;
    String phone;
    String email;

    public LookupByUrlParamControllerNew () { }

    // Main method that takes the incoming parameter and first checks accounts, bef
ore checking contacts
    public string redirectToCustomer() {

        Map<String,String> params = ApexPages.currentPage().getParameters();
        if(params.size() > 0) {
            customerName = params.get('customer_name');
            customerNumber = params.get('customer_number');
            phone = params.get('phone');
            email =  params.get('email');
        }

        String customerUrl;
        customerUrl = redirectToAccount();
        if ( customerUrl == 'null')
        {
            customerUrl = redirectToContact();
        }

        return customerUrl;

    }

    public string redirectToAccount() {

        Account account;
        // Check the Customer Name
        try {
            if(customerName != null) {
                account = [select ID from Account where name = :customerName limit
1];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
           account = null;
        }

        // Check the Account Number
        try {
            if(customerNumber != null) {
                account = [select ID from Account where AccountNumber = :customerNu
mber limit 1];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
            account = null;
        }

        // Check the Main Phone Number (no US format check)
        try {
            if ((account == null) && (phone != null)) {
                 account = [select ID from Account where phone = :phone limit 1];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
            account = null;
        }

        // Check the Main Phone Number (including US format check)
        try {
            if ((account == null) && (phone != null)) {

                String npa;
                String nnx;
                String extension;

                //  Added logic for NA mobile phone numbers
                if (phone.length() == 10) {
                    npa = phone.substring(0,3);
                    nnx = phone.substring(3,6);
                    extension = phone.substring(6,10);
                    phone = '(' + npa + ') ' + nnx + '-' + extension;
                }

                account = [select ID from Account where phone = :phone limit 1];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
            account = null;
        }

        // Return the acccount reference if a match is found. Otherwise return null

        String accountUrl = 'null';
        if(account != null) {
            accountUrl = '/' + account.Id;
        }

        return accountUrl;
    }


    public string redirectToContact() {

        Contact contact = null;

        // Check the Customer Name
        try {
            if(customerName != null) {
                contact = [select ID from Contact where Name = :customerName limit
1];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
            contact = null;
        }
        // Check the Main Phone Number (no US format check)
        try {
            if ((contact == null) && (phone != null)) {
                 contact = [select ID from Contact where Phone = :phone limit 1];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
            contact = null;
        }

        // Check the Mobile Phone Number (no US format check)
        try {
            if ((contact == null) && (phone != null)) {
                 contact = [select ID from Contact where MobilePhone = :phone limit
 1];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
            contact = null;
        }

        // Check the Home Phone Number (no US format check)
        try {
            if ((contact == null) && (phone != null)) {
                 contact = [select ID from Contact where HomePhone = :phone limit 1
];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
            contact = null;
        }

        // Check the Other Phone Number (no US format check)
        try {
            if ((contact == null) && (phone != null)) {
                contact = [select ID from Contact where OtherPhone = :phone limit 1
];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
            contact = null;
        }

        // Check the Main Phone Number (including US format check)
        try {
            if ((contact == null) && (phone != null)) {

                String npa;
                String nnx;
                String extension;

                //  Added logic for NA mobile phone numbers
                if (phone.length() == 10) {
                    npa = phone.substring(0,3);
                    nnx = phone.substring(3,6);
                    extension = phone.substring(6,10);
                    phone = '(' + npa + ') ' + nnx + '-' + extension;
                }

                contact = [select ID from Contact where phone = :phone limit 1];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
            contact = null;
        }

        // Check the Mobile Number (including US format check)
        try {
            if ((contact == null) && (phone != null)) {
                String npa;
                String nnx;
                String extension;

                //  Added logic for NA mobile phone numbers
                if (phone.length() == 10) {
                    npa = phone.substring(0,3);
                    nnx = phone.substring(3,6);
                    extension = phone.substring(6,10);
                    phone = '(' + npa + ') ' + nnx + '-' + extension;
                }

                contact = [select ID from Contact where MobilePhone = :phone limit
1];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
            contact = null;
        }

        // Check the Home Number (including US format check)
        try {
            if ((contact == null) && (phone != null)) {
                String npa;
                String nnx;
                String extension;

                //  Added logic for NA mobile phone numbers
                if (phone.length() == 10) {
                    npa = phone.substring(0,3);
                    nnx = phone.substring(3,6);
                    extension = phone.substring(6,10);
                    phone = '(' + npa + ') ' + nnx + '-' + extension;
                }

                contact = [select ID from Contact where HomePhone = :phone limit 1]
;
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
```

```
            contact = null;
        }

        // Check the Other Number (including US format check)
        try {
            if ((contact == null) && (phone != null)) {
                String npa;
                String nnx;
                String extension;

                //  Added logic for NA mobile phone numbers
                if (phone.length() == 10) {
                    npa = phone.substring(0,3);
                    nnx = phone.substring(3,6);
                    extension = phone.substring(6,10);
                    phone = '(' + npa + ') ' + nnx + '-' + extension;
                }

                contact = [select ID from Contact where OtherPhone = :phone limit 1
];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty account page
            contact = null;
        }

        // Check the email address
        try {
            if ((contact == null) && (email != null)) {
                contact = [select ID from Contact where Email = :email limit 1];
            }
        } catch (System.queryException e) {//no entry found for lookup item, displa
y empty contact page
            contact = null;
        }

        // Return the Contact ID if found, otherwise return the handle to create a
new contact.
        String contactUrl;
        if(contact != null) {
            contactUrl = '/' + contact.Id;
        }
        else {
            contactUrl = '/003/e';
        }

        return contactUrl;
    }

    // Following are a number of test methods for this class
    //
    public static testMethod void testLookupByUrlParamAccount() {
        LookupByUrlParamControllerNew controller = new LookupByUrlParamControllerNe
w();
        controller.customerName = 'Avaya';
        String redirectUrl = controller.redirectToCustomer();
        System.assertEquals(redirectUrl, '/003/e');
    }

    public static testMethod void testLookupByUrlParamInvalidAccount() {
        LookupByUrlParamControllerNew controller = new LookupByUrlParamControllerNe
w();
        controller.CustomerName = '';
        String redirectUrl = controller.redirectToCustomer();
        System.assertEquals(redirectUrl, '/003/e');
    }

     public static testMethod void testLookupByUrlParamInvalidContact() {
        LookupByUrlParamControllerNew controller = new LookupByUrlParamControllerNe
w();
        controller.CustomerName = '';
        String redirectUrl = controller.redirectToCustomer();
        System.assertEquals(redirectUrl, '/003/e');
    }

    public static testMethod void testLookupByUrlParamPhone() {
        LookupByUrlParamControllerNew controller = new LookupByUrlParamControllerNe
w();
        controller.phone = '1234';
        String redirectUrl = controller.redirectToCustomer();
        System.assertEquals(redirectUrl, '/003/e');
    }

    public static testMethod void testLookupByUrlParamWherePhoneNumberIs10Chars() {
        LookupByUrlParamControllerNew controller = new LookupByUrlParamControllerNe
w();
        controller.phone = '1234567891';
        String redirectUrl = controller.redirectToCustomer();
        System.assertEquals(redirectUrl, '/003/e');//no record found
    }


    public static testMethod void testLookupByUrlParamWhereVallidPhoneNumberIs10Cha
rs() {
        LookupByUrlParamControllerNew controller = new LookupByUrlParamControllerNe
w();
        controller.phone = '2125551234';
        String redirectUrl = controller.redirectToCustomer();
        System.assertEquals(redirectUrl, '/003/e');//no record found
    }

    public static testMethod void testLookupByUrlParamInvalidPhoneNumber() {
        LookupByUrlParamControllerNew controller = new LookupByUrlParamControllerNe
w();
        controller.phone = '';
        String redirectUrl = controller.redirectToCustomer();
        System.assertEquals(redirectUrl, '/003/e');//no record found
    }

    public static testMethod void testLookupByUrlParamAccountNumber() {
        LookupByUrlParamControllerNew controller = new LookupByUrlParamControllerNe
w();
        controller.customerNumber = '4321';
        String redirectUrl = controller.redirectToCustomer();
        System.assertEquals(redirectUrl, '/003/e');
    }

    public static testMethod void testLookupByUrlParam() {
        LookupByUrlParamControllerNew controller = new LookupByUrlParamControllerNe
w();
        String redirectUrl = controller.redirectToCustomer();
        System.assertEquals(redirectUrl, '/003/e');
    }
}
```

APPENDIX 3 – LOOKUPBYPARM64 SOURCE CODE

```
public class LookupByUrlParamController64 {

    public String customerName;
    public String phone;
    public String customerNumber;
    public String email;


    public LookupByUrlParamController64 () { }

    // Main method that takes the incoming parameters and first checks accounts, be
fore checking contacts
    public string redirectToCustomer() {
        String customerUrl;


        Map<String,String> params = ApexPages.currentPage().getParameters();

        if(params.size() > 0) {
            customerName = params.get('customer_name');
            customerNumber = params.get('customer_number');
            phone = params.get('phone');
            email =  params.get('email');

            //Call method to check for an account matching these parameters
            customerUrl = redirectToAccount();

            if ( customerUrl == 'null')
            {
                //No account match check for a contact
                customerUrl = redirectToContact();
            }

        }
        return customerUrl;

    }

    public string redirectToAccount(){
        String query;
        Account account;
        Boolean paramAdded = False;

        //assemble the database query based on the passed in parameters
        query = 'SELECT ID FROM Account WHERE ';// WHERE (ID is not null ';

        // Check the Customer Name
        if(customerName != null) {

            query += ' name like \'%' + customerName + '%\'';
            paramAdded = True;

        }

        // Check the Account Number
        if(customerNumber != null) {

            if(paramAdded)
            {
              query += ' AND';
            }
            query += '  AccountNumber = \''+ customerNumber + '\'';
            paramAdded = True;

        }

        // Check the Phone field in original or US format
        if (phone != null) {

            if (USFormatPhone(phone) != null)
            {
                if(paramAdded)
              {
                query += ' AND';
              }
                query += ' (phone = \'' + phone  +'\''+
                                +' OR phone = \'' + USFormatPhone(phone) +'\''+

                                +') ';
                paramAdded = True;
            }
            else
            {
                if(paramAdded)
              {
                query += ' AND';
              }
                query += ' (phone = \'' + phone + '\') ';
                paramAdded = True;
            }
        }

        // Run the compiled query if params added
        if (paramAdded) {
          try {

             query += ' LIMIT 1';
             account = Database.query(query);

          } catch (System.queryException e) {//no entry found for lookup item, disp
lay empty account page
              //system.debug(e.getMessage());
              account = null;
          }
        }
        // Return the Account reference if a match is found. Otherwise return null

        String accountUrl = 'null';

        if(account != null) {
            accountUrl = '/' + account.Id;

        }

        return accountUrl;
    }

    //Function to search for contacts matching the passed in parameters
    public string redirectToContact(){
        //Declare local variables
        String contactQuery;
        Contact contact = null;
        Boolean paramAdded = False;

        //assemble query based on parameters passed
        contactQuery = 'select ID from Contact WHERE';


        // Check the Phone fields in original or US format
        if (phone != null) {
            if (USFormatPhone(phone) != null)
            {
                if(paramAdded)
              {
                contactQuery += ' AND';
              }
                contactQuery += ' (phone = \'' + phone +'\''+
                                +' OR phone = \'' + USFormatPhone(phone) +'\''+
                                +' OR MobilePhone = \'' + phone +'\''+
                                +' OR MobilePhone = \'' + USFormatPhone(phone) +'\'
'+
                                +' OR HomePhone = \'' + phone +'\''+
                                +' OR HomePhone = \'' + USFormatPhone(phone) +'\''+
                                +' OR OtherPhone = \'' + phone    +'\''+

                                +' OR OtherPhone = \'' + USFormatPhone(phone) +'\''
+
                                +') ';
                paramAdded = True;
            }
            else //the phone number could not be parsed into a US format so just th
e original format checked
            {
                if(paramAdded)
              {
                contactQuery += ' AND';
              }
                contactQuery += ' (phone = \'' + phone +'\''+
                                + ' OR MobilePhone = \'' + phone +'\''+
                                + ' OR HomePhone = \'' + phone +'\''+
                                + ' OR OtherPhone = \'' + phone +'\''+
                                + ') ';
                paramAdded = True;
            }
        }

        // Check the email address
        if (email != null) {
            if(paramAdded)
            {
              contactQuery += ' AND';
            }
            contactQuery += ' Email = \'' + email + '\'';
            paramAdded = True;
        }

        //execute query if params added
        if (paramAdded)
        {
          try
          {

              //Limit results to one record
              contactQuery += ' LIMIT 1';
             //execute the created SQL
             contact = Database.query(contactQuery);

          } catch (System.queryException e) {//no entry found for lookup item, disp
lay empty account page
              //system.debug(e.getMessage());
              contact = null;
          }
        }

        // Return the Contact ID if found, otherwise return the handle to create a
new contact.
        String contactUrl;
        if(contact != null) {

            contactUrl = '/' + contact.Id;

        }
        else {

            contactUrl = '/003/e';
        }

        return contactUrl;

    }

    public string USFormatPhone(String lphone) {
        //This function is used to parse the phone number passed and format into th
e US phone format for searching

        //Declare variables
        String npa;
        String nnx;
        String extension;
        String rphone;

        //  Added logic for NA mobile phone numbers
        if (lphone.length() == 10) {
            npa = lphone.substring(0,3);
            nnx = lphone.substring(3,6);
            extension = lphone.substring(6,10);
            rphone = '(' + npa + ') ' + nnx + '-' + extension;
        }

        //return the parsed phone number to the search function
        return rphone;
    }
}
```

APPENDIX 4 – APEX CLASS OTHER SAMPLES

This section contains a number of sample Apex Classes which can be adopted to work in an Enterprise Salesforce Account.

global class AccountCustomerPriority {

```
  webservice String area;
  webservice String region;

  //Define an object in apex that is exposed in apex web service
  global class CustomerPriority {
    webservice String priority;
  }

  webservice static CustomerPriority getAccountPriority(String accountID) {

      SObject result = [select  CustomerPriority__c from account where Id = :accountID limit 1];
      CustomerPriority cp = new CustomerPriority();
      cp.priority = (String)result.get('CustomerPriority__c');
      return cp;
```

This above code returns a customer priority value to Avaya Contact Center based on customer account number ID.

https://na7.salesforce.com/apex/LookupByUrlParam?email=%VALUE%

```
global class AccountCustomerPriorityByEmail {

  webservice String area;
  webservice String region;

  //Define an object in apex that is exposed in apex web service
  global class CustomerPriority { webservice String priority;
  }

  webservice static CustomerPriority getAccountPriority(String email) {

      SObject result = null;
      CustomerPriority cp = new CustomerPriority();

      try {
          result = [select  CustomerPriority__c from account where email__c = :email limit 1];
          cp.priority = (String)result.get('CustomerPriority__c');
      } catch(Exception e) {
          cp.priority = 'Low';
      }

      return cp;
  }
```

This above code returns a customer priority value to Avaya Contact Center based on customer email address.

END OF DOCUMENT