NICE inContact DEVone Partner Quick Start Guide

Version 1

December 2017

Central 3
Download Software 3
Create a Campaign 3
Create a Phone Skill 4
Add a Phone Call Contact 4
Test Your Script 5
Create a Station 5
Create Unavailable Codes 5
Create a Team 6
Create an Agent Account 6
Developer 8
Introduction 8
NICE inContact API Framework 8
Studio 9
OAuth2 Authentication Overview 10
OAuth2 Authentication Step-By-Step 11
Cross-Origin Resource Sharing Support 24
Appendix A 25

Central

Download Software

Overview

The following instructions demonstrate how to download the primary set of files you will commonly use, such as Studio.Application.

1. Go to http://login.incontact.com. Log in with the administrative username and password listed above. You should change this password as soon as possible by clicking **Admin > Users** and selecting your username.
2. Click **Admin > Support > Software Update** to download the primary set of files you will commonly use, such as Studio.Application. The executable files are the actual programs, not the installation files. Most find it convenient to create a folder (i.e., 'inContact') on their hard drive and save the files there. You can save the files directly to your desktop, but many people don't like the additional clutter these files generate. Your same account number will provide access to each of these programs (although if you create new users, they will only have the permissions you assign them).

   After installation, the programs will download additional support files as necessary. Once you have downloaded your tools, you can then access the **Routing** menu in Central to create a campaign, skills, dispositions, etc.

   Note:
   Use Internet Explorer when downloading the Studio application.

Create a Campaign

Overview

Campaigns allow you to group related skills for reporting purposes. The typical intent is that a service bureau would create a separate campaign for each client. If you want to generate reports by department, you could create campaigns for departments within your organization.

This topic explains how to create a campaign.

1. Click **Routing > Campaigns**.
2. Click **Create New** to display the new campaign wizard.
3. Enter a value in the **Campaign Name** field.
4. Click **Create Campaign**.

Create a Phone Skill

Overview

Every contact handled by inContact must be assigned a skill, whether or not a live agent is involved. Before you can assign a phone number, you must create a skill that can be tied to the number (a skill belongs to a campaign, so a campaign must be created first).

This topic explains how to create a phone skill.

1. In Central, click **Routing >Skills**.
2. Click **Create New >Single Skill**.
3. Select ***Phone Call*** for **Media Type**.
4. Enter a unique value in **Skill Name**.
5. Select a campaign.
6. Select ***Inbound*** or ***Outbound*** for **Inbound/Outbound**.
7. Complete any other applicable fields for the skill.
8. Click **Create Skill**.

Add a Phone Call Contact

Overview

Phone numbers must be provisioned by the inContact Support team before they are available. Once a number is provisioned, you can associate the number with a script.

This topic instructs you on how to create a point of contact.

1. In Central, click **Routing >Points of Contact**.
2. Click **Create New**. A new window opens.
3. From the **Media Type** drop-down list, select ***Phone Call***.
4. Select one of the phone numbers for **Contact DNIS**.
5. From the **Default Skill** drop-down list, select a skill.
6. From the **Script** drop-down list, select the script you previously created.
7. Click **Create Point of Contact**.

Test Your Script

Overview

This topic instructs you on how to test your script and add a request agent action.

1. Dial the phone number you selected. If the script was built correctly, you will hear hold music.
2. To accommodate agents into your solution, add stations and agents (who also have assigned skills). In a script call flow, include a **REQAGENT** (request agent) action to queue the caller. When an agent is available, their phone will ring.

Create a Station

Overview

Station is the NICE inContact term for a physical device where an agent communicates with contacts, such as a phone or workstation. When agents log in to their agent application, they enter the station ID assigned to the device they are using.

This topic explains how to create a new station.

1. Click **Admin > Users > Stations**.
2. Click **Create New**.
3. Enter the **Station Name**, the **Phone Number**, and the **Caller ID** you want to display when this caller makes outbound calls. Keep in mind that the station name represents the specific telephone associated with the agent's location. In most cases, it is confusing to name the station after the agent. (For example, "John Smith's Desktop" is a more precise name than "John Smith").
4. Click **Create Station**.
5. Take note of the new station ID that is generated.

Create Unavailable Codes

Overview

For first-time account setups, you can create unavailable codes (also called outstates). Unavailable codes are reasons that agents can select for why they are unavailable to handle calls. The most common unavailable codes include break, training, lunch, meeting, and wrap. Wrap means "wrap-up time" which is defined as the time an agent needs to handle a call after the actual call has ended. Wrap-up time may include activities such as documenting the call, sending a fax, researching an issue, etc.

Unavailable codes are not required. If unavailable codes have not been configured for an agent’s team, then the agent will need to click Unavailable to go unavailable. In most cases, however, unavailable codes provide very useful reporting information regarding how the agent spends his/her time while unavailable.

This topic explains how to create unavailable codes.

1. Click **Admin >Users > Unavailable Codes**.
2. Click **Create New**.
3. Enter the **Unavailable Code Name**.
4. If you want to include information for **After Contact Work**, click the **Post Contact** checkbox.
5. Click **Create Unavailable Code**.
6. Repeat this process for each of the unavailable codes that you want to create.

Create a Team

Overview

A default team is included, but you may want to create a new team (or rename the default team). A team is defined for reporting purposes only. In most cases, a supervisor oversees a group (or team) of agents. A team name might be “Customer Service”, “Customer Service (Morning)”, “Customer Service (John)”, etc.

This topic explains how to create a team.

1. Click **Admin>Users >Teams**.
2. Click **Create New**.
3. Enter the **Team Name.**
4. Click **Create Team**.

Create an Agent Account

Overview

This topic explains how to create an agent account.

1. Click **Admin>Users >Users**.
2. Click **Create New**.
3. From the **Team** drop-down list, select the team you want this agent to belong to.

4. From the **Security Profile** drop-down list, select the profile you want to assign to this agent. A profile is a group of rights and privileges that determines the screens and reports an agent can access. There are many default profiles you can select: ***Agent***, ***Manager***, ***Scripter***, ***Super Scripter***, and ***Supervisor***. If you don't want to use any of these default profiles, you can create your own. Refer to help.incontact.com for more information on profiles.
5. Enter the agent's **First Name**, **Last Name**, **Username**, **Email Address**, and **Password**.
6. Click **Create User**.

CHANGE AN AGENT'S PASSWORD

1. Click **Admin>Users >Users**.
2. Select the agent's name.
3. Click **General**.
4. Click **RESET PASSWORD** . NICE-inContact will email the agent a new password.

Developer

Introduction

There are two basic methods for interacting with the NICE inContact system using Application Programming Interfaces (APIs):

- The NICE inContact API Framework
- Studio

Both of these methods are described below. The remainder of this guide details how to get started with the NICE inContact API Framework. For more information on our APIs, refer to our online help located at help.incontact.com.

NICE inContact API Framework

The NICE inContact API Framework is a collection of RESTful APIs that provide access to NICE inContact data and services. All RESTful API requests are made over HTTPS. Each request requires a valid API token, which is retrieved via an OAuth2 authentication process. API tokens expire, and can be renewed. Because the RESTful API works over HTTPS, it can be used from any platform, programming language, or environment that supports HTTPS (pretty much every language and environment there is).

API calls are grouped in our documentation based on their common use or function. The following API groups currently exist:

• Admin

• Agent

• Real-time Data

• Patron Services

Additional APIs are planned for future releases.

In addition to our RESTful APIs, NICE inContact distributes SDKs that are focused on particular types of solutions. SDKs contain guides, documentation, libraries, and sample code that aid in efficiently creating solutions. The following SDK currently exists:

• Agent SDK

Additional SDKs are constantly being developed, and will be made available as they are completed.

The RESTful APIs and SDKs are described below.

Admin API

The Admin API is a collection of API calls that deal with system objects in the NICE inContact system. For instance, the Admin API will allow you to change skill assignments for agents. The Admin API also includes some admin-level functions, such as recording or monitoring a call, and remotely logging off an agent. This API is not yet complete, and mostly provides access to data that can be used to augment dashboards or agent applications. Future releases of the Admin API will provide full access to all NICE inContact system objects and relationships.

Agent API

The Agent API is a collection of API calls that deal with an authenticated agent session. Through the Agent API, you can create applications that log agents in, allow them to control their status, and enable them to make and receive calls and other contacts. The Agent API currently supports the voice, work item, and chat channels. Support for email and voicemail will be added in later releases.

Real-Time Data API

The Real-Time Data API is a collection of API calls that provide access to real-time data on the platform. This API can be used to create custom dashboards and control panels, and to provide information to agents. The Real-Time data API should not be used to access data sets that span more than 30 days. Also, you may only access data that is no more than two years old.

Patron Services API

The Patron Services API is a collection of API calls that can be used to create patron-facing applications. For example, you can:

• Create a mobile application that allows a patron to request a callback at a future time.

• Create a website that allows users to request a live chat session with an agent.

• Create applications or services that create work items to be routed in the NICE inContact system (such as routing trouble tickets or sales opportunities from a web request, or from a process that is monitoring a CRM system).

Agent SDK

The Agent SDK shows how to use the Agent API to create fully-functional agent applications. A guide is provided that describes important concepts on the NICE inContact platform. A sample HTML application is also provided that shows how to use the Agent API from JavaScript. The Agent API can be used to create stand-alone agent applications, or to embed agent functionality in other applications (like CRMs). For example, the NICE inContact Agent for Salesforce was developed using the Agent API.

Studio

Studio is a Windows application that allows you to create workflow scripts that run on the NICE inContact platform. Interactive Voice Response (IVR) scripts can be created, as well as generic workflow scripts. Scripts can access SOAP-based and RESTful APIs on other systems. Scripts can be scheduled to run continuously, or at regular intervals on the NICE inContact platform. For example, you can create scripts that guide a caller through an IVR tree, monitor an external system (such as a CRM) for work to be routed, and provide agent functionality. All of the RESTful API calls can also be used from a workflow script. You could, for example, use the Real-Time Data API methods to create a script that regularly pulls real-time data from NICE inContact and feeds it to an external dashboard system. You could also use the Agent API methods to create an IVR-driven script that provides agent functionality (for agents who don't use a computer, and only have a phone).

Workflow scripts are powerful applications that run on the NICE inContact platform.

OAuth2 Authentication Overview

To use any of the NICE inContact APIs, you must have a current and valid token. Tokens are generated by the NICE inContact Token Service. The NICE inContact Token Service is an OAuth 2.0 based service, and follows the OAuth 2.0 specification. In version 2.0 of the NICE inContact API Framework, only the Implicit, Password, and Client OAuth grant types are supported. Support for the Authorization Code grant type will be added in a future release.

Each supported OAuth grant type is designed for optimal security and usability for applications of a specific type:

• **Implicit:** best for browser-based applications that run on the client (i.e. JavaScript applications, or similar). They can also be used for web applications that run on a server.

• **Password:** best for applications that are not web-based.

• **Client:** best for applications that do not need to access resources for a specific authenticated user.

Implicit and Password tokens are required when making API calls that require a user context. For example, retrieving a list of agents from the platform requires a user context, and the API call will fail if the user specified does not have the required permissions (even if the user name and password are valid).

Implicit Tokens

Implicit tokens are best used for web applications that run in a browser (such as JavaScript applications). They can also be used for web applications that run on a web server, and that are accessed through a browser. Implicit tokens contain information about a specific user. The user is directed by your application to a special NICE inContact authentication website, where they enter their NICE inContact credentials. When your application browses to the NICE inContact website, it also sends a requested redirect URI to which you would like the NICE inContact website to redirect the user after they successfully authenticate. The requested redirect URI must match one of the redirect URIs you specify when you register the application with NICE inContact (see below). The API token is sent by NICE inContact to the redirect URI (which also makes it available to the browser-based application). Your application can use this token directly to make API calls. The token identifies the application, the vendor, and the authenticated user. Any API calls made with the token will be limited by the user's permissions within NICE inContact, and also by the limited scope of the application, as configured in the application registration.

Password Tokens

If your application is server-based (but not running on a web server), or desktop-based (but not running in a browser), or a native mobile application, and if it needs access to resources on behalf of a specific user, you should use the Password grant type. The user credentials used to create a password token are either known by the application, or provided to the application by the user. Your application uses the username and password to request and receive an API token directly from the NICE inContact authorization server. Password tokens contain information about the authenticated user. The user is authenticated by sending the user name and password along with the token request. If the user name and password are valid, a token is granted. The token identifies the application, the vendor, and the user. Any API calls made with the token will be limited by the user's permissions within NICE inContact, and also by the limited scope of the application, as configured in the application registration.

Client Tokens

Client tokens are best used in browser-based, server-based, desktop-based, and mobile applications that do not need to identify a specific user. You do not send user name and password information with client token requests. Client tokens are useful for API calls that do not require a user context. For example, if you are creating a web page that allows a patron (caller) to request a web chat with an agent, you would use the "Start Chat" API call. This call does not require a user context, since the patron making the request is not a user in the NICE inContact system. As the API calls are being made through a browser in an uncontrolled environment, the token could be visible to any other applications running in the environment. Although you could use an Implicit or Password token to use this API, using a Client token is more secure, since it is more limited in its scope. The request and resulting Client token are not usable for API calls that require a user context, so if a rogue or malicious application running in the browser happens to "steal" the Client token, it won't be able to do much with it.

Once the application has a token generated by the Token Service, it must send the token with each API call it makes. The token allows the NICE inContact API Framework to identify the application, the application vendor, and the user who is using the application (if applicable).

OAuth2 Authentication Step-By-Step

In order for the Token Service to encode the identity of the application, vendor, and user in the token, this information must be provided when you request the token. You must register the vendor and application names through the application registration process in the NICE inContact Central website (**Manage > API Applications**). When you register your application, the Registration Service will generate an Application ID for your application. This confidential ID can be used with the application and vendor names to retrieve a token using the Authorization Code grant type (which is not currently supported). Do not use the Application ID (a.k.a. client secret) with the Implicit, Password, or Client grant types. Instead, use the Business Unit Number.

Note:

The Business Unit Number – together with your vendor and application names – can be used to retrieve a token that is linked to you and your application.

The vendor and application names are combined with the Business Unit Number to create an authorization key, which you must send to the Token Service when you request a token. The authorization key is a Base64-encoded string that is comprised of the vendor and application names, and the Business Unit Number:

```
AppName@VendorName:BUNum
```

After Base64-encoding this string and sending it to the Token Service, you will receive a token that you can use with any NICE inContact API. Password and Client tokens are generally set to expire after an hour. Implicit tokens last for 60 days. When a token has expired, using the token with an API call will generate an error from the NICE inContact API Framework. The error will identify that the token has expired. Your application must then re-request a new token. This can be done by sending the authorization key again (for the Password and Client grant types), or by using a refresh token to request a new API token (for the Implicit grant type). The refresh token is provided with the API token when the API token is initially granted. Your application can store the refresh token for use when it is notified by the NICE inContact API Framework that your API token has expired. You can use the refresh token, or request a new token using the same method you retrieved the original token. Either method will generate a new API token and refresh token. Which method you choose will depend on the type of application you have created, and the user experience you want your users to have.

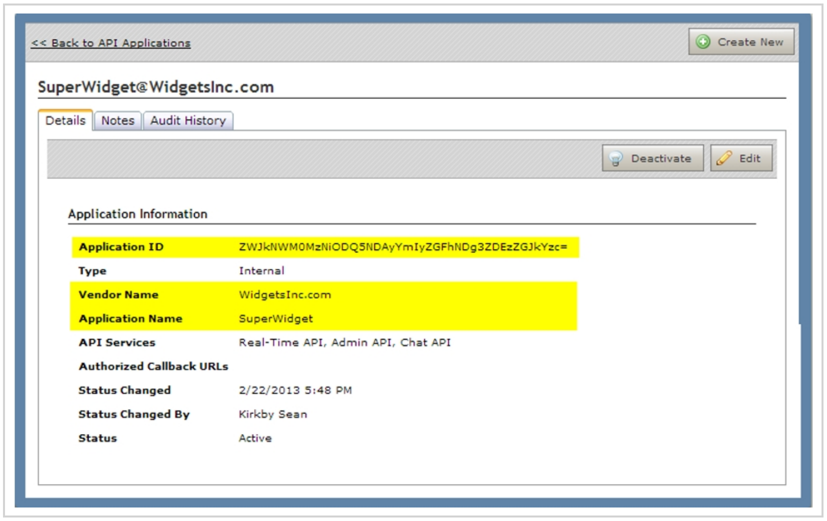
If your application uses the Implicit grant type, the user will be directed to the NICE inContact website to enter their credentials to identify themselves and to authorize your application to access their NICE inContact account. When an Implicit token expires, you can go through this process again, redirecting the user to the NICE inContact website again, and prompting them for their credentials again. If you do not wish to force your user to do this again when the API token expires, you can use the refresh token to retrieve a new API token.

The next section will show you how to retrieve a token for use with the NICE inContact APIs, how to use the token with the APIs, and how to retrieve a new API token once the original token expires.

OAUTH2 AUTHENTICATION PROCESS

1. **Register your application in the NICE inContact Central website.**
   In order to retrieve an API token, you must register your application. You do this in the NICE inContact Central website (**Manage > API Applications**.) If you do not see this menu option, your security profile is likely preventing you from seeing it. Contact your NICE inContact administrator, or consult NICE inContact technical support for assistance.

   When you register your application, you must specify a **Vendor Name** and an **Application Name**. You must also specify which of the available APIs your application is authorized to use. The **Vendor Name**, **Application Name**, and **Business Unit Number** will be used to retrieve a valid and current API token.



SuperWidget@WidgetsInc.com

Application Information

Application ID	ZWJkNWM0MzNiODQ5NDAyYmIyZGFhNDg3ZDEzZGJkYzc=
Type	Internal
Vendor Name	WidgetsInc.com
Application Name	SuperWidget
API Services	Real-Time API, Admin API, Chat API
Authorized Callback URLs	
Status Changed	2/22/2013 5:48 PM
Status Changed By	Kirkby Sean
Status	Active

2. **Create an authorization key.**
   In order to retrieve a token, you must send an authorization key to the NICE inContact token service. The authorization key is the Base64-encoded value of a string that is comprised of the **Vendor Name**, **Application Name**, and the **Business Unit Number** for your application. Refer to Step1:

   ```
   AppName@VendorName:BUNum
   ```

   For example, if you register your application with a Vendor Name of "WidgetsInc.com", and an Application Name of "SuperWidget", and if your Business Unit Number is "123456", the authorization key would be the Base64 encoding of the following string:

   ```
   SuperWidget@WidgetsInc.com:123456
   ```

   The Base64 encoded version of this string is:

   ```
   U3VwZXJXaWRnZXRAV2lkZ2V0c0luYy5jb206MTIzNDU2
   ```

   You provide the Application and Vendor names when you register your application with NICE inContact. You can always look them up, if needed, in the **Manage > API Applications** section of the NICE inContact Central website. If you have not yet registered your application, you will need to do so in this section of the Central website. Refer to Step 1.

3. **Send a token request.**
   You must decide which token type you will use in your application: Implicit, Password or Client.
   If your application is a web-server-based application, or a browser-based application, and if it needs access to resources on behalf of a specific user, you should use the Implicit grant type.

   If your application is server-based (but not running on a web server), or desktop-based (but not running in a browser), or a native mobile application, and if it needs access to resources on behalf of a specific user, you should use the Password grant type.

   If your application needs access to NICE inContact resources that are not owned by a specific user, regardless of whether it is server-based, browser-based, desktop-based, or mobile, you should use the Client grant type.

   Implicit Tokens
   Implicit tokens contain information about a specific user. The user is sent by your application to a special NICE inContact authentication website, where they enter their credentials. You also send a requested redirect URI that you would like the NICE inContact website to redirect the user to after they successfully authenticate. The requested redirect URI must match one of the redirect URIs you specified when you registered the application with NICE inContact. The API token is sent by NICE inContact to the redirect URI (which also makes it available to the browser-based application). Your application can use this token directly to make API calls. The token identifies the application, the vendor, and the user. Any API calls made with the token will be limited by the user's permissions within NICE inContact, and also by the limited scope of the application, as configured in the application registration.

   Password Tokens
   Password tokens also contain information about a specific user. The user is authenticated by sending the user name and password along with the token request. If the user name and password are valid, a token is granted. The token identifies the application, the vendor, and the user. Any API calls made with the token will be limited by the user's permissions within NICE inContact, and also by the limited scope of the application, as configured in the application registration.

   Implicit and Password tokens are required when making API calls that require a user context. For example, retrieving a list of agents from the platform requires a user context, and the API call will fail if the user specified does not have the required permissions (even if the user name and password are valid).

   Client Tokens
   Client tokens do not contain information about a user. You do not send user name and password information with client token requests. Client tokens are useful for API calls that do not require a user context. For example, if you are creating a web page that allows a patron/caller to request a web chat with an agent, you would use the **Start Chat** API call. This call does not require a user context, since the patron making the request is not a user in the NICE inContact system. As the API calls are being made through a browser in an uncontrolled environment, the token could be visible to any other applications running

in the environment. Although you could use an Implicit or Password token to use this API, using a Client token is more secure, since it is more limited in its scope. The request and resulting Client token are not usable for API calls that require a user context, so if a rogue or malicious application running in the browser happens to steal the Client token, it won't be able to do much with it.

The request must be sent to the NICE inContact Token Service URL:

https://api.incontact.com/InContactAuthorizationServer/Token

The HTTPS request must have an **Authorization** header. The Authorization header must be included using the following syntax:

```
Authorization: basic {code}
```

For instance, using the Base64-encoded authorization code from the example above, you would create an Authorization header like this:

```
Authorization: basic
U3VwZXJXaWRnZXRAV2lkZ2V0c0luYy5jb206MTIzNDU2
```

You must also include a request payload in the body of the request. The payload is a JSON object that identifies the supported OAuth grant type you selected (Refer to Step 2), and that provides any other information needed by the selected grant type.

If you selected the Password grant type, the request body should look like this:

```
{
"grant_type":"password", "username":"myuser@mydomain.com",
"password":"myP@ssw0rd", "scope":"Admin Chat RealTime"
}
```

Note:

The **grant_type** for the Password token request should be set to ***password***. Also note that the **Username** and **Password** are required parameters for this grant type. The **Scope** parameter identifies the APIs for which the token can be used. For version 2.0 of the NICE inContact API Framework, the following APIs can be specified:

- Admin
- Chat
- RealTime
- Agent

Any of these APIs can be used with a password token. If you want to retrieve a Client token, the request body should look like this:

```
{
"grant_type":"client_credentials", "scope":"Chat"
}
```

Note:

The **grant_type** for the Client token request is ***client_credentials***. Also note that since no user is required for the Chat API, client tokens can be used with this API. The Admin, RealTime and Agent APIs require a user context, so client tokens cannot be used with these APIs.

Below are examples of HTTP requests and responses for each of these grant types:

```
// TODO: section for "Implicit" grant REQUEST/RESPONSE goes
here
```

Request URL: `https://ap-i.incontact.com/InContactAuthorizationServer/Token`

Request Method: `POST`

Request Headers:

```
POST /InContactAuthorizationServer/Token HTTP/1.1
```

```
Host: api.incontact.com

Connection: keep-alive

Content-Length: 98

Origin: http://myapp.example.com

Authorization: basic
U3VwZXJXaWRnZXRAV2lkZ2V0c0luYy5jb206MTIzNDU2

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWe-
bKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.95 Safar-
i/537.11

Content-Type: application/json; charset=UTF-8

 Accept: application/json, text/javascript, */*; q=0.01

Referer: http://myapp.example.com/mainPage

Accept-Encoding: gzip,deflate,sdch

Accept-Language: en-US,en;q=0.8

 Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

Request Payload:

```
{

"grant_type":"password",

"username":"myuser@mydomain.com",

"password":"myP@ssw0rd",

"scope":"Admin Chat RealTime"

}
```

“Password” grant RESPONSE:

Response Status Code: 200 Ok

Response Body:

```
{
```

```
"access_
token":"VAp4BUKosVW3DQa6mGVbF9gJVOCoi7fv7+Kp8piexdF24JjuwWey-
i/7jpU2YZVMwHIpfefuDsLFF14 4iMsJJeghS0uDMb+XO1wToISr8UswnFm-
lWDPiI2JuC5OlBLoRLNiG1nWfxkvf2xg4vZ3TC5w5856I1u8oQsVx9j43x
VEZJsuKR2TC3y-
dGNfk-
p+/xWu8ad67kRDcnqmV+/vi6fYVWwJWaeXO6jFxJd8TYCVaYka1rf5uO065VKSuG++cq2I7y
 2zws/XEu8vwC95ZREXOr4UcgNqs+IIfl4IdNEqCBM=", "token_
type":"bearer",

"expires_in":3600,

"refresh_token":"3s2Q+ul6VUujc/SpFjOSoQ==",

"scope":"Admin Chat RealTime",

"resource_server_base_uri":"ht-
tps://api.incontact.com/InContactAPI/",

"refresh_token_server_uri":"https://api.incontact.com/
InContactAuthorizationServer/Token"

}
```

“Client” grant REQUEST:

Request URL: https://ap-
i.incontact.com/InContactAuthorizationServer/Token

Request Method: POST

Request Headers:

```
POST /InContactAuthorizationServer/Token HTTP/1.1

 Host: api.incontact.com

Connection: keep-alive

Content-Length: 65

Origin: http://myapp.example.com
```

```
Authorization: basic
U3VwZXJXaWRnZXRAV2lkZ2V0c0luYy5jb206MTIzNDU2

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWe-
bKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.95 Safar-
i/537.11

Content-Type: application/json; charset=UTF-8

Accept: application/json, text/javascript, */*; q=0.01

Referer: http://myapp.example.com/mainPage

Accept-Encoding: gzip,deflate,sdch

Accept-Language: en-US,en;q=0.8

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

Request Payload:

```
{
"grant_type":"client_credentials",
"scope":"Admin Chat RealTime"
}
```

“Client” grant RESPONSE:

Response Status Code: 200 Ok

Response Body:

```
{
"access_
token":"VAp4BUKosVW3DQa6mGVbF9gJVOCoi7fv7+Kp8piexdF24JjuwWey-
i/7jpU2YZVMwyzfyrjpT30NiMT
D72kZi5Pj8Kw/vuEVZgh2JYgnPXgQ+EjfkHve-
heT47XUNgx1Kb7rqJeQrxNLkbx20T/E8UDw==",
"token_type":"bearer",
"expires_in":3600,
"refresh_token":null,

"scope":"Admin Chat RealTime",

"resource_server_base_uri":"ht-
tps://api.inContact.com/InContactAPI/",

"refresh_token_server_uri":"https://api.inContact.com/

InContactAuthorizationServer/Token"

}
```

Other than different **grant_type** values in the request payload (and the inclusion of ***User-name*** and ***Password*** parameters for the password flow), the requests are identical.

4. **Retrieve the token.**
   The token service returns a JSON object in the response body that contains the access token, as well as other information that you will need. For example, the following could be returned as part of the Password grant response:

   `Response Body:`

```
{

"access_
token":"VAp4BUKosVW3DQa6mGVbF9gJVOCoi7fv7+Kp8piexdF24JjuwWey-
i/7jpU2YZVMwHIpfefuDsLFF14 4iMsJJeghS0uDMb+XO1wToISr8UswnFm-
lWDPiI2JuC5OlBLoRLNiG1nWfxkvf2xg4vZ3TC5w5856I1u8oQsVx9j43x
VEZJsuKR2TC3y-
dGNfk-
p+/xWu8ad67kRDcnqmV+/vi6fYVWwJWaeXO6jFxJd8TYCVaYka1rf5uO065VKSuG++cq2I7y
 2zws/XEu8vwC95ZREXOr4UcgNqs+IIfl4IdNEqCBM=", "token_
type":"bearer",

"expires_in":3600,

"refresh_token":"3s2Q+ul6VUujc/SpFjOSoQ==",

"scope":"Admin Chat RealTime",

"resource_server_base_uri":"https://api-c4.in-
contact.com/InContactAPI/",

"refresh_token_server_uri":"https://api-c4.incontact.com/

InContactAuthorizationServer/Token"

}
```

   The response body contains a JSON object that provides the token, the expiration time limit for the token, the refresh token, the scope for the token (i.e. which APIs the token can be used with), and a resource server base URI and refresh token server base URI. Each of these is explained below:

   `access_token:` `this is the token value you send with each API call (see below for details)`

   `token_type:` `this is the value included before the token in the Authorization header for each API call (see below for details)`

   `expires_in:` `this is the number of seconds after which the token will expire (from the time of the token's creation)`

refresh_token: this is the token used to retrieve a new API token when the API token expires

resource_server_base_uri: this is the base URI for accessing resources, based on the application and user identifiers supplied in the authorization key. You must use this base URI to use any of the APIs with the access_token.

refresh_token_server_uri: this is the base URI for the token service used to retrieve a new token when the original token expires, using the refresh token.

5. **Use the token.**
   Once you get the token response, you can use the token in any API call. All of the NICE inContact API calls are RESTful, and require an Authorization header with a valid token:

   ```
   Authorization: bearer {token}
   ```

   For example, if you want to use the Admin API to retrieve a list of agents in your business unit, you would send an HTTPS request to the Agent List resource URI, and include the API token as a bearer Authorization header:

   ```
   "Agents" API REQUEST:


   Request URL: https://ap-
   i.incontact.com/inContactAPI/services/v1.0/agents


   Request Method: GET


   Request Headers:


   GET /inContactAPI/services/v1.0/agents HTTP/1.1

    Host: api.incontact.com

   Connection: keep-alive  Content-Length: 65

   Origin: http://myapp.example.com

   Authorization: bearer
   VAp4BUKosVW3DQa6mGVbF9gJVOCoi7fv7+Kp8piexdF24JjuwWey-
   i/7jpU2YZVMwHIpfefuDsLFF 144iMsJJeghS0uDMb+XO1wToISr8UswnFm-
   lWDPiI2JuC5OlBLoRLNiG1nWfxkvf2xg4vZ3TC5w5856I1u8oQsVx9j43xVE
   ZJsuKR2TC3y-
   dGNfk-
   p+/xWu8ad67kRDcnqmV+/vi6fYVWwJWaeXO6jFxJd8TYCVaYka1rf5uO065VKSuG++cq2I7y2zws
    Eu8vwC95ZREXOr4UcgNqs+IIfl4IdNEqCBM=

   User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWe-
   bKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.95 Safar-
   i/537.11

   Content-Type: application/json; charset=UTF-8

   Accept: application/json, text/javascript, */*; q=0.01
   ```

```
Referer: http://myapp.example.com/mainPage

Accept-Encoding: gzip,deflate,sdch

Accept-Language: en-US,en;q=0.8

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

6. **Expired tokens.**
   When you retrieve a token from the Token Service, the token response indicates how long the token will last before it expires. This is typically one hour.

   If you attempt to use a token that is expired in an API call, the API service will return an HTTPS status of "401 Unauthorized". If you get this response, simply request a new token using the same method that you used to retrieve the original token, and send the API request again with the new token.

Cross-Origin Resource Sharing Support

When writing applications that run in browsers, such as a JavaScript application, it is a typical use case to store JavaScript code on a server in one domain, but have the code access resources that belong to another domain.

Unfortunately, this approach is also used by malware developers. To mitigate the ability of malware from one domain, running in an unsuspecting user's browser, to access and abuse resources that belong to another domain, most web browsers prevent cross-domain scripting or cross-domain resource access. The browser's JavaScript detects such attempts and blocks them.

However, there are legitimate use cases where JavaScript code on a server in one domain needs to access resources that reside in another domain. A number of methods can be used to make this possible, despite the browser's official stance to disallow it.

One standards-based approach is known as Cross-Origin Resource Sharing, or CORS. The CORS standard allows the server that owns a resource to have a say in whether or not code from another domain can access its resources. The following URLs provide good tutorials and introductions to CORS support in various browsers.

http://enable-cors.org/

http://www.html5rocks.com/en/tutorials/cors/

The NICE inContact RESTful APIs can be used from browsers that support CORS. This includes the latest versions of Chrome, Firefox, and Safari, as well as IE 10.

CORS Support in Modern Browsers

There are two types of CORS requests: simple, and not-so-simple.

Simple CORS requests are HTTP GET requests that only contain text/plain responses.

Not-so-simple CORS requests are requests that use other HTTP methods, or that receive data in other formats (such as Content-type: application/json). Not-so-simple CORS requests will typically result in 2 HTTP requests for each API call. The browser will send a preflight OPTIONS request to verify that the HTTP request being proposed is allowed by the server. Once confirmed by the NICE inContact API, the browser will then issue the proposed request. This is a function of the browser. You should include the appropriate headers to cause the browser to cache the preflight response in order to reduce the number of requests. Note that preflight responses will only be cached for identical requests.

Note:

HTTP error responses are typically not visible through XHR requests. The CORS implementations of most modern browsers only return 200 OK responses. Other responses have a status code of 0, and no status description. The NICE inContact API Framework will include the status code and description of error statuses in the response body, so your code can react to error conditions properly.

Appendix A

JavaScript (with jQuery) sample code for getting an API token:

- Code Sample 1 uses a username and password to get a password token.
- Code Sample 2 uses an implicit token, where you redirect the user to the NICE inContact login page, and NICE inContact sends a token back to your browser app.
- Code Sample 3 shows how to use the refresh token to get a new token if the old one expires.

Note:

You only have to use the refresh token if you used an Implicit grant request to get a token, and you don't want to force the user to log in again when the token expires. If you use the Password grant request, and your token expires, just re-request the token again (using the same method you used the first time – see Code Sample 1).

Getting a Password Token (Javascript)

```
var authCode = window.btoa("appName@vendor.example.com:999999999");
var result ={};
$.ajax({
    url:"https://api.incontact.com/inContactAuthorizationServer/Token",
    type:"POST",
    contentType:"application/json",
    dataType: "json",
8     data: JSON.stringify({
9         "grant_type":"password",
10         "username":"inContact_user@example.com",
11         "password":"p@ssw0rd!",
12         "scope":"Agent Admin Chat RealTime"
13     }),
14     headers:{
15         "Authorization": "basic "+ authCode
16     },
17     success: function (data) {
18         result.access_token = data.access_token;
19         result.token_type = data.token_type;
20         result.expires_in = data.expires_in;
21         result.refresh_token = data.refresh_token;
22         result.scope = data.scope;
23         result.resource_server_base_uri = data.resource_server_base_
uri;
24         result.refresh_token_server_uri = data.refresh_token_server_
uri;
25     },
26     error: function (xhr, status) {
27         alert(xhr.status);
28     }
29 });
```

Line 1

>>Use the application name, vendor name, and the business unit number from your application registration in the Central website. The window.btoa() method base64encodes the string. This method may not be supported in all browsers. If it's not supported in your target browser, find another way to base64-encode the string. The format of the string is shown: the application name, followed by "@", followed by the vendor name, followed by ":", followed by the business unit number. This auth code is used in the token request (see line 15).

Line 6

>>Make sure to set the content type to application/json! You will get an empty 200 OK response if you don't.

Lines 8-13

>>Use JSON.stringify to turn the token request object into a JSON string for the request body. The fields in the token request object are shown in these lines. Use a valid NICE inContact username and password. The scope field indicates what access you are requesting for the token (the APIs the token can be used to call). The scope should be limited to the minimum access you request. The token will be limited to the access you request, or to the access allowed by the application registration (whichever is more restrictive).

Lines 14-16

>>Add an authorization header that uses the auth code that is created in line 1.

Lines 17-25

>>If the request is successful, the return data is a JSON object with the access token and other information encoded. Note especially the token itself (line 18), and the resource_server_base_uri (line 23). You should use the resource_server_base_uri for every API call, and add the token as an Authorization header (of type bearer).

Getting an Implicit Token (Javascript)

```
var result = {};
var implicitTokenUri =
    "https://api.inContact.com/InContactAuthorizationServer/Authenticate";
var client_id = "appName@vendor.example.com";
var token_scope = "RealTime Admin Chat Agent"
var redirect_uri = document.URL;
var state_object = "myState";

function RedirectToAuthPage() {
    var url = implicitTokenUri;
     url = url + "?client_id=" + encodeURIComponent(client_id);
     url = url + "&redirect_uri=" + encodeURIComponent(redirect_uri);
     url = url + "&response_type=token";
     url = url + "&scope=" + encodeURIComponent(token_scope);
     url = url + "&state=" + state_object;
     window.location.href = url;
}

$(document).ready(function () {
     var query_string = {};
     var query = window.location.search.substring(1);
     var vars = query.split("&");
     for (var i=0;i<vars.length;i++) {
         var pair = vars[i].split("=");
         query_string[pair[0]] = pair[1];
     }
       if (typeof(query_string.access_token) != "undefined") {
         tokenResponse.access_token = query_string.access_token;
         tokenResponse.token_type = query_string.token_type;
         tokenResponse.expires_in = query_string.expires_in;
         tokenResponse.refresh_token = query_string.refresh_token;
         tokenResponse.refresh_token_server_uri = query_string.refresh_token_server_uri;
         tokenResponse.scope = query_string.scope;
         tokenResponse.resource_server_base_uri = query_string.resource_server_base_uri;      }
});
```

Line 2

>>Note that the URI for an Implicit grant request is different from the password grant request!

Lines 3-4

>>The client ID is the application name and vendor name (separated by "@"). Note that there is no business unit number or client secret. Note also that the client ID *is not base64-encoded* for implicit grants! The scope field indicates what access you are requesting for the token (the APIs the token can be used to call). The scope should be limited to the minimum access you request. The token will be limited to the access you request, or to the access allowed by the application registration (whichever is more restrictive).

Line 5

>>When the user successfully provides credentials to the NICE inContact login page, the login page will redirect the browser to the URL you specify. Note that this URL must be added to the application registration in the Central website. Otherwise, the NICE inContact login page will present an error (401).

Line 6

>>You can pass a state object (string) to the login page. Upon successful login, the login page will pass the state object back to the redirect URL. This allows you to keep continuity between the initial page before you pass control to NICE inContact, and the redirect page after the NICE inContact login page.

Lines 8-16

>>Call this method to redirect to the login page. Pass the client ID, redirect URL, scope, and state objects as query parameters. Note also the response_type. For implicit grant requests, it should be token.

Lines 18-36

>>This jQuery call assigns an anonymous function to the document.ready event. When the HTML document is fully loaded, this method will run. This method takes the query string and parses it into an object. It then attempts to access properties of this object to save them into the result object. You could parse directly into result, but this code sample gives you a chance to see what fields you should expect when the page is loaded. Of course, this method is run when your application first loads it, so the method checks to see if access_token was one of the parameters (which it will be when the NICE inContact login page redirects to it). It assumes that if access_token exists in the query string, then all of the other parameters will be there also.

Line 27

>>Use this token in an Authorization header with each API call.

Line 29-31

>>The expires_in value indicates (in seconds) how long the token will last before you need a new one. This is typically 60 days (5,184,000 seconds). When the token expires, you could choose to send the user back to the inContact login page to get a new token. Or, you can use the refresh token. Send the refresh token to the refresh_token_server_uri, and a new token will be returned (Refer to Code Sample 3, below).

Line 32

>>This field specifies the API scope of the token (what APIs the token can be used to call).

Line 33

>>The resource_server_base_uri is important! You must make API requests to this base URI. Append the relative URI of any given API call to this base URI. This base URI is a cluster-specific URI for the user's business unit.

Using the Refresh Token (Javascript)

```
1 var authCode = window.btoa("appName@vendor.example.com:999999999");
2 var result = {};
3 $.ajax({
4     url: result.refresh_token_server_uri;
5     type: "POST",
6     contentType: "application/json",
7     dataType: "json",
8     data: JSON.stringify({
9         "grant_type": "refresh_token",
12         "refresh_token": result.refresh_token
13     }),
14     headers: {
15         "Authorization": "basic " + authCode

16     },
17     success: function (data) {
18         result.access_token = data.access_token;
19         result.token_type = data.token_type;
20         result.expires_in = data.expires_in;
21         result.refresh_token = data.refresh_token;
22         result.scope = data.scope;
23         result.resource_server_base_uri = data.resource_server_base_
uri;
24         result.refresh_token_server_uri = data.refresh_token_server_
uri;
25     },
26     error: function (xhr, status) {
27         alert(xhr.status);
28     }
29 });
```

Line 4, 8-13

>>Using the refresh token to request a new token is very similar to the Password grant request method. The only real difference is the request URL, and the request body. Use the URL and the refresh token that were returned with the original token request.