# Rapid Development Framework (RDF)

## For Amazon Connect

User Guide 1.2

May 2023

# Revision History

While reasonable efforts have been made to ensure that the information in this document is complete and accurate, CompuCom reserves the right to make changes and corrections without the obligation to notify any person or organization of such changes. Such changes and corrections will be reflected in this Revision History table.

| VERSION | DATE | DESCRIPTION | AUTHOR |
|---|---|---|---|
| 0.1 | Apr 18, 2023 | Draft version. | Julius Malixi |
| 1.0 | Apr 20, 2023 | Updates made from initial internal CompuCom review. | Julius Malixi |
| 1.1 | Apr 28, 2023 | Ready for internal CompuCom walkthrough. | Julius Malixi |
| 1.2 | May 4, 2023 | Updates made from internal CompuCom walkthrough. Ready for customer review. | Julius Malixi |

# Table of Contents


Revision History ..... 2

Chapter 1: Overview ..... 5

Chapter 2: Getting Started ..... 6

Prerequisites ..... 6

RDF Account Number ..... 6

Contact Flow Set Up ..... 6

Queue Set Up ..... 7

Add/Update Business Hours ..... 9

Add/Update Agent Statuses ..... 9

Create Cross-Account IAM Role ..... 9

Create Cross-Account KMS Key ..... 9

Create Application S3 Bucket ..... 10

Create DynamoDB Tables (Optional) ..... 12

Chapter 3: Hello World! ..... 13

Chapter 4: Application Configuration ..... 15

config.csv ..... 15

<tenant>-callflow.csv ..... 18

<tenant>-tts.csv ..... 19

<tenant>-holidays.csv ..... 20

<tenant>-sysadmin.csv ..... 20

Chapter 5: Application Development ..... 21

CALL FLOWS ..... 21

BLOCK ..... 21

TYPE ..... 21

PARAM ..... 21

ACTION ..... 21

Session Attributes ..... 22

Python Built-in Data Types ..... 22

BLOCK TYPES ..... 23

PLAY ..... 23

MENU ..... 24

INPUT ..... 26

LANGUAGE ..... 28

SUB ..... 29

**EXEC** ..... 30
**EVAL** ..... 31
**DNIS** ..... 31
**CASE** ..... 32
**USERDATA** ..... 33
**SETQUEUE** ..... 34
**QUEUE** ..... 34
**XFER** ..... 35
**REST** ..... 35
**SOAP** ..... 36
**RECORD** ..... 38
Chapter 6: System Administration ..... 41
**Broadcast Messages** ..... 41
**Dynamic Menus** ..... 42
Chapter 7: RDF Built-In Functions ..... 43
**Check Business Hours** ..... 43
**Check Holiday** ..... 43
**Queue Metrics** ..... 43
**DynamoDB** ..... 44
**Clear Previous Menu** ..... 45
**Contact Dispositions** ..... 45
Chapter 8: Helpful Hints ..... 46
**RDF Lambda Cache** ..... 46
**Menu Tracker** ..... 47
**Queue Tracker** ..... 47
**Contact Attributes** ..... 47
**Pointing to DEV/UAT/PROD** ..... 48
**Quick Connects** ..... 49
**Agent Whispers** ..... 49
**CloudWatch Monitoring** ..... 50
Chapter 9: Sample Application ..... 51
Chapter 10: Support ..... 62

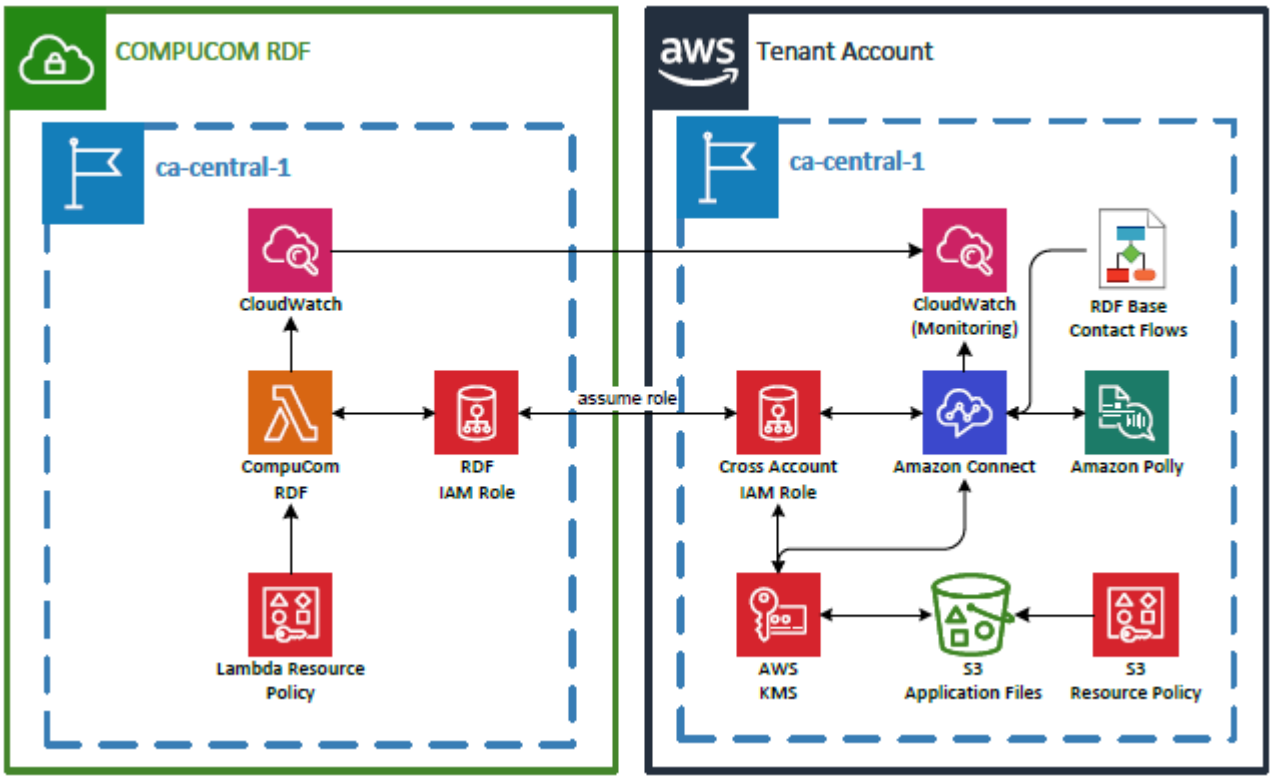
# Chapter 1: Overview

The CompuCom Rapid Development Framework (RDF) *("the framework")* is a flexible and simple configuration tool used to assist in the rapid development of Interactive Voice Response (IVR) and Queue applications on Amazon Connect. The framework was developed to overcome limitations in Amazon Connect related to Multilingual Support, Call Flow Labels, single prompts within Play and Input blocks, Computations and Scripting, Global and Local Error Handling, IVR Caller Dispositions, Contact Attribute and Data Persistence across modules, DNIS Routing and Integration with backend applications such as SOAP and REST web services.

Amazon Connect provides a great range of features to implement a Contact Center of any size, and the CompuCom RDF tool combines these capabilities into a development framework that delivers robust applications using building blocks normally found in traditional IVR Integrated Development Environments (IDE).

The CompuCom RDF tool was developed using Python and provides developers with simple access to the Python scripting language enabling developers to quickly perform computational and conditional operations within their applications without having to continually break out into custom lambda functions. Seamless integration to other AWS Cloud Services, Web Services, and external applications can be easily customized.

The CompuCom RDF architecture comprises of several pre-built Amazon Connect Contact Flows (RDF Base Contact Flows) that interacts with a Lambda function using cross-account IAM roles hosted on a remote AWS account. It also leverages an S3 Bucket to store application configuration and audio files and a cross-account KMS Key to access these S3 resources. Optional features include DynamoDB tables. Below is a high-level architecture diagram of the CompuCom *Rapid Development Framework*:



# Chapter 2: Getting Started

This section describes the initial steps required to get the CompuCom RDF environment ready on Amazon Connect.

## Prerequisites

This User Guide assumes that the user has a thorough understanding of IVR application development, a working knowledge of Amazon Connect, and knows how to create an AWS Account and Amazon Connect Instance.

- ➔ To create a new AWS Account: https://docs.aws.amazon.com/controltower/latest/userguide/provision-and-manage-accounts.html
- ➔ To create an Amazon Connect Instance: https://docs.aws.amazon.com/connect/latest/adminguide/amazon-connect-instances.html
- ➔ To claim a Phone Number: https://docs.aws.amazon.com/connect/latest/adminguide/tutorial1-claim-phone-number.html

Before getting started, download all the CompuCom RDF GetStarted files to a local folder:

- ➔ git clone https://github.com/CompuComRDF/GetStarted

Once downloaded, you should see the following folders:

```
[ec2-user@ip-172-31-27-110 GetStarted]$ ls -l
total 0
drwxrwxr-x 2 ec2-user ec2-user 167 May  3 13:52 ContactFlows
drwxrwxr-x 4 ec2-user ec2-user  35 May  3 13:52 SampleApplication
drwxrwxr-x 2 ec2-user ec2-user  46 May  3 13:52 UserGuide
[ec2-user@ip-172-31-27-110 GetStarted]$
```

## RDF Account Number

- ➔ Take note of the RDF AWS Account Number used for your Amazon Connect instance as this will be needed when configuring your AWS tenant account: **037338416846**

## Contact Flow Set Up

Once you have your Amazon Connect instance and Phone Number set up, and the CompuCom GetStarted files downloaded, go to the ContactFlows folder.

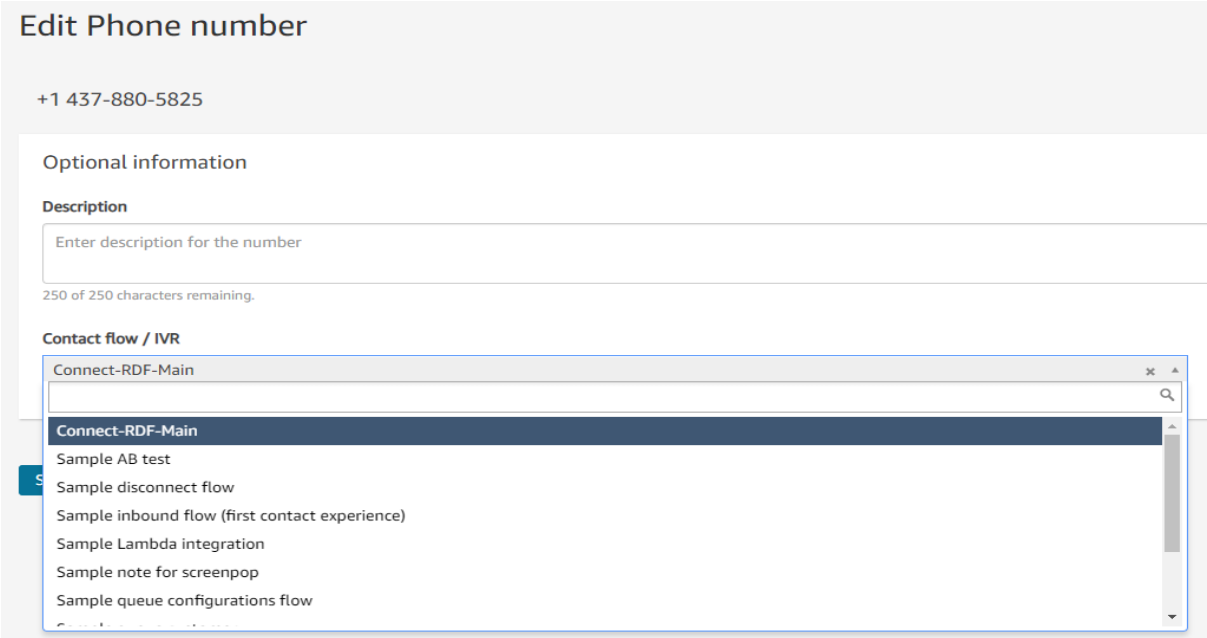
- ➔ The **ContactFlows** folder will contain the following Base Contact Flows required by CompuCom RDF:

| Filename | Contact Flow Type |
|---|---|
| Default customer whisper | Default customer whisper |
| Connect-RDF-Outbound | Outbound whisper |
| Connect-RDF-Queue | Customer queue |
| Connect-RDF-Whisper | Agent whisper |
| Connect-RDF-Agent | Transfer to queue |
| Connect-RDF-Main | Contact flow |

➔ Import and publish the above Contact Flows to your Amazon Connect instance, ensuring you select the correct Contact Flow Type. For **Connect-RDF-Outbound**, you will need to assign a claimed Phone Number from your Connect instance.

➔ **IMPORTANT:** Import and publish the Contact Flows ***in the order listed in the above table.*** This will ensure that any settings dependent from another Contact Flow are published and available in the correct order. For additional details on how to import Amazon Contact flows, refer to the following: https://docs.aws.amazon.com/connect/latest/adminguide/contact-flow-import-export.html

➔ Assign the Phone Number(s) to use the **Connect-RDF-Main** Contact Flow:

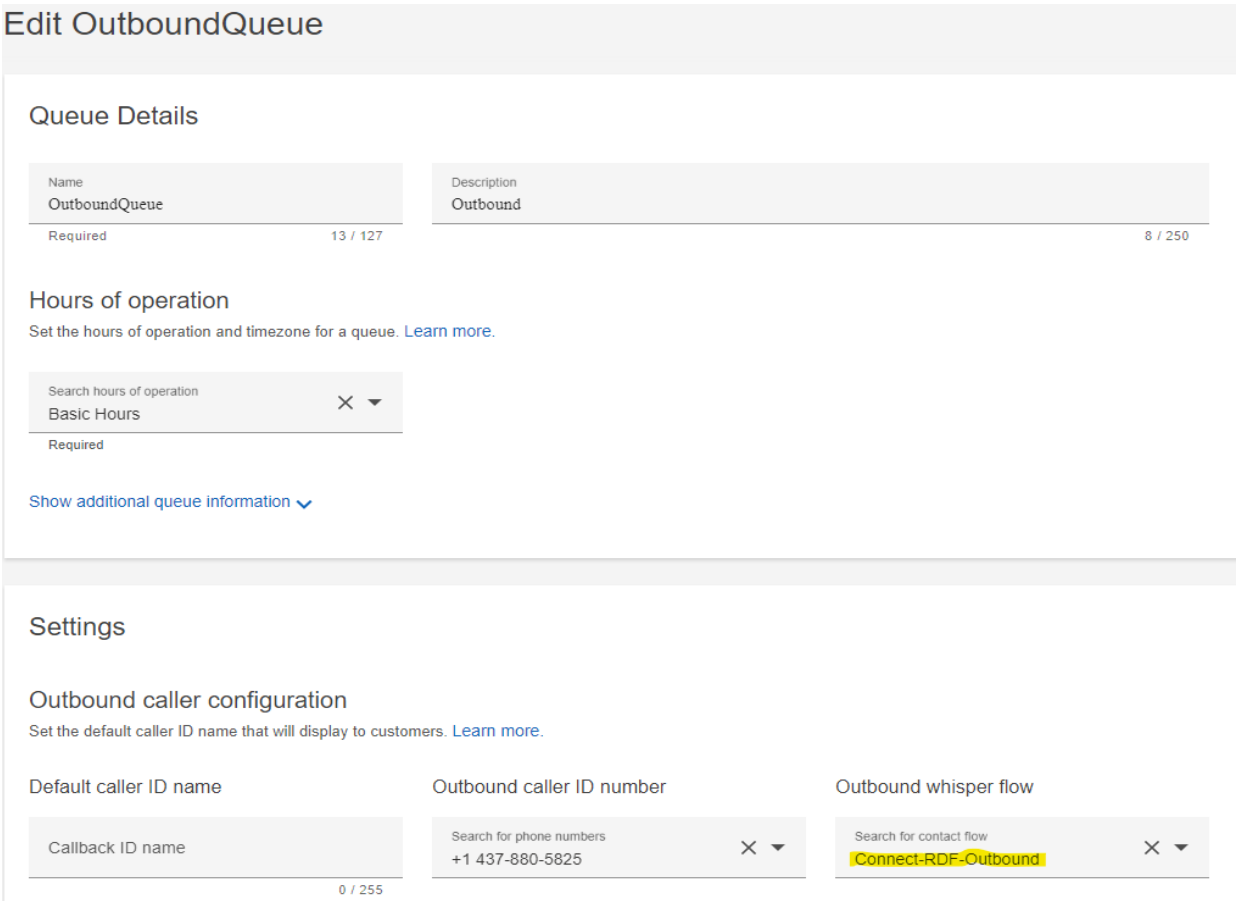



## Queue Set Up

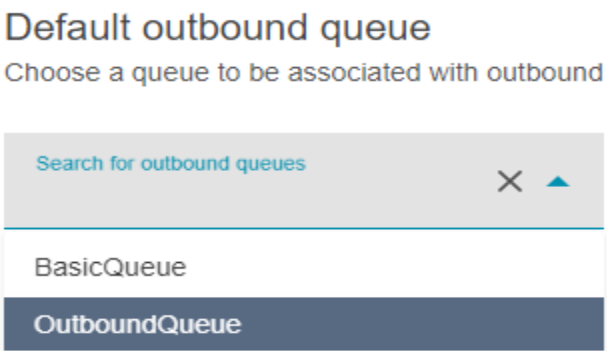
➔ If you already have a list of queues needed for your application, create all the queues: https://docs.aws.amazon.com/connect/latest/adminguide/create-queue.html

You will need the Queue **Name's** when referencing a queue in the SETQUEUE block explained in Chapter 5.

➔ Create an **OutboundQueue** and assign the **Connect-RDF-Outbound** Contact Flow and make sure to input a Callback ID name and select the correct Outbound caller ID number you want to use for your instance:




➔ Update the **Basic Routing Profile** and select the **OutboundQueue** created in the previous step as the Default outbound queue.




➔ Create other Routing Profiles as required ensuring you select the **OutboundQueue** as the Default outbound queue:
https://docs.aws.amazon.com/connect/latest/adminguide/routing-profiles.html

## Add/Update Business Hours

➔ Add/Update your Business Hours schedule as needed:
https://docs.aws.amazon.com/connect/latest/adminguide/set-hours-operation.html

## Add/Update Agent Statuses

➔ Add/Update custom agent statuses:
https://docs.aws.amazon.com/connect/latest/adminguide/agent-custom.html

## Create Cross-Account IAM Role

Follow these steps to create a cross-account IAM role on your AWS account to allow the remote RDF lambda function to assume this role:

➔ Go to **IAM -> Roles** and click on **Create role.**

➔ Under **"Select trusted entity"**, select **AWS account** as the Trusted entity type.

➔ Under **"An AWS account"**, choose **Another AWS account** and enter the RDF Account Number for your Amazon Connect instance (see: RDF Account Number), and click **Next**.

➔ Under **"Add permissions"**

➔ In the Filter policies box, type Connect.
In the list of Amazon Connect policies, select **AmazonConnect_FullAccess**

➔ In the Filter policies box, type DynamoDB (optional).
In the list of DynamoDB policies, select **AmazonDynamoDBFullAccess**

➔ For Role name, type **xaccount-rdf-role** (this must be the exact name used for this role)

➔ Choose **Create role.**

## Create Cross-Account KMS Key

Follow these steps to create a cross-account KMS Key to be used for the S3 Bucket that is used to store application configuration and audio files:

➔ Go to **Key Management Service** -> **Customer managed keys** -> **Create Key** -> Choose Symmetric, Encrypt and decrypt -> Under **"Advanced options"** select **KMS** and **Single-Region key** and click **Next**.

➔ Under **“Add Labels”**, enter Alias -> **xaccount-s3-rdf-kms** and click **Next**.

➔ Under **“Define key administrative permissions”**, select an appropriate Key administrator from the list of IAM users and roles and check **“Allow key administrators to delete this key”**, then click Next.

➔ Under **“Define key usage permissions”**, scroll down to **“Other AWS accounts”** and click **“Add AWS Account”**. Enter the RDF account number to use this Key -> RDF Account Number, then click **Next**.

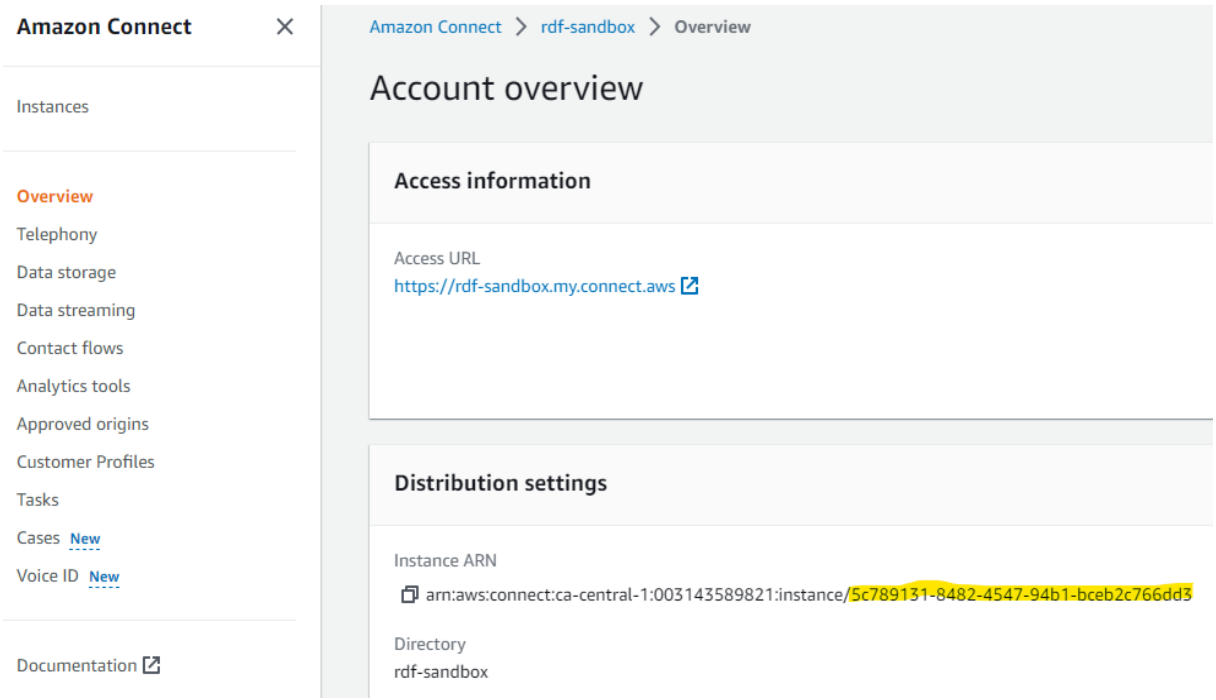
➔ Under **“Review”**, add the following entry to the KMS Key Policy for **connect.amazonaws.com**:

```
{
   "Sid": "Enable Amazon Connect",
   "Effect": "Allow",
   "Principal": {
      "Service": "connect.amazonaws.com"
   },
   "Action": "kms:decrypt",
   "Resource": "*"
},
```

## Create Application S3 Bucket

Follow these steps to create an S3 Bucket used to store application configuration and audio files:

➔ Before creating your S3 Bucket, go to Amazon Connect and take note of the Amazon Connect Instance ID (as highlighted in yellow below):




➔ Go to the **S3** Console and select **Create bucket**.

➔ For Bucket Name, use the Amazon Connect Instance ID prefixed with **"amazon-connect-"** as follows: ***s3://amazon-connect-5c789131-8482-4547-94b1-bceb2c766dd3***

(Replace the Instance ID in the example above with your Connect Instance ID)

➔ Under **"Default encryption"**, choose **"AWS Key Management Service key (SSE-KMS)"** as the Encryption key type, select **"Choose from your AWS KMS Keys"** and from the drop-down list, select the KMS Key **xaccount-s3-rdf-kms** that was created from the previous step.

➔ Click **Create Bucket** and then go to your S3 Bucket's Permissions.

➔ Edit your **Bucket Policy** and add the following entry to provide RDF and Amazon Connect with S3 access to this bucket (update only the highlighted areas of this policy with your S3 bucket name, Amazon Connect ARN and the tenant AWS Account Number):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::037338416846:role/service-role/RDF-role-k0m3ejei"
      },
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3::: amazon-connect-5c789131-8482-4547-94b1-bceb2c766dd3/*"
    },
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Principal": {
        "Service": "connect.amazonaws.com"
      },
      "Action": [
        "s3:ListBucket",
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3::: amazon-connect-5c789131-8482-4547-94b1-bceb2c766dd3",
        "arn:aws:s3::: amazon-connect-5c789131-8482-4547-94b1-bceb2c766dd3/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:connect:ca-central-1:003143589821:instance/5c789131-8482-4547-94b1-bceb2c766dd3",
          "aws:SourceAccount": "003143589821"
        }
      }
    }
  ]
}
```

# Create DynamoDB Tables (Optional)

If your application requires DynamoDB tables, create your tables by following these instructions:

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/getting-started-step-1.html

# Chapter 3: Hello World!

This chapter walks you through a sample Hello World application that will get your Amazon Connect instance to begin processing calls with the CompuCom RDF. This sample application provides sample code of the common features used in the framework such as different Play Prompts, Menu's, Broadcast Messages, Dynamic Menus, Web Services, and System Administration.

➔ From the CompuCom GetStarted files downloaded in your local folder, go to the SampleApplication/config folder and you will find all the csv configuration files required by a tenant application as shown in the table below. This sample application can also be used as a starting point for other tenant applications. Simply replace the name **"sample"** with a tenant name.

| Filename | Description |
|---|---|
| config.csv | Instance Configuration File |
| sample-callflow.csv | Call Flow Configuration File |
| sample-tts-csv | TTS/SSML Configuration File |
| sample-sysadmin | System Administration Configuration File |
| sample-holidays.csv | Holidays Configuration File |
| sysadmin-callflow.csv | System Administration Module (for demonstration only) |
| ws-callflow.csv | Web Services Demo Module (for demonstration only) |

➔ Go to the **SampleApplication/** folder and upload both the **config/** and **prompts/** sub folders to the root folder of the S3 bucket created in Chapter 2 – Create Application S3 Bucket.

```
[ec2-user@ip-172-31-27-110 SampleApplication]$ ls -l
total 0
drwxrwxr-x 2 ec2-user ec2-user 197 Apr 18 18:23 config
drwxrwxr-x 3 ec2-user ec2-user  20 Apr 18 18:23 prompts
```

➔ Your Application S3 Bucket should have the following folder structure:

- ***config/***
- ***prompts/sample/en-US***
- ***prompts/sample/fr-CA***

➔ The **prompts/sample/** folder will contain the standard audio wav files for the different languages supported:

| Prompts Folder | Description |
|---|---|
| en-US/ | English Audio Folder containing default audio files |
| fr-CA/ | French Audio Folder containing default audio files |

➔ **NOTE:** The **"sample"** sub-folder is used for demonstration purposes only. In a real-world application, this name will be replaced by a tenant name. This can be useful for multiple tenants to share the same Amazon Connect instance.

➔ Send a request to CompuCom with your **AWS tenant name** and **Amazon Connect Instance ID** to provide proper permissions for your Instance to invoke the RDF lambda function.

➔ Make a test call to your Amazon Connect phone number and follow the prompts. Refer to Chapter 9 – Sample Application to review the sample call flow diagrams.

# Chapter 4: Application Configuration

This chapter describes how to create a tenant application and what configuration files are used by the CompuCom RDF. All the files outlined in this section are the configuration files needed to control your Amazon Connect application. In other words, there is no need to update any of the base RDF Contact Flows in Amazon Connect. You can simply upload these application configuration files to the S3 Bucket and the changes will take effect immediately. See section Chapter 8 – RDF Lambda Cache to understand how caching has been implemented on the framework.

## config.csv

The **config.csv** file is a tenant specific configuration file used to define your application's default settings. This is the initial configuration file that the CompuCom RDF uses that defines all other configuration files used in your application such as Callflow(s), TTS, Holiday, and System Administration files. All other parameters defined in this file control the global behaviour of your application for error handling and global treatment of DTMF keys. These global settings alleviate the need to continually define repetitive settings throughout your call flow. However, these settings can also be overridden at anytime in your call flow as needed. For example, in the configuration sample below, since we set option 9 globally to return a caller to a previous menu, there is no need to define option 9 elsewhere in the application unless you want option 9 to override the global behaviour at a specific menu. This also applies to option * used for repeat.

| | A | B |
|---|---|---|
| 1 | KEY | VALUE |
| 2 | tenant | sample |
| 3 | callFlow | sample-callflow.csv |
| 4 | ttsFile | sample-tts.csv |
| 5 | holidayFile | sample-holidays.csv |
| 6 | sysAdminFile | sample-sysadmin.csv |
| 7 | connectFlowType | INBOUND |
| 8 | defaultDNIS | IVR100 |
| 9 | defaultQueue | BasicQueue |
| 10 | language | en-US |
| 11 | globalMenuAttempts | 2 |
| 12 | globalInvalidPrompt | tts:InvalidInput1\|tts:InvalidInput2 |
| 13 | globalTimeoutPrompt | tts:NoInput1\|tts:NoInput2 |
| 14 | globalMaxtriesPrompt | tts:MaxInputXfer |
| 15 | globalMaxtriesAction | PQ100 |
| 16 | globalMenuRepeat | * |
| 17 | globalMenuPrevious | 9 |
| 18 | globalZeroPrompt | agent |
| 19 | globalInputTerm | # |
| 20 | speechAnalytics | off |
| 21 | chatAnalytics | off |

The following table provides a description of all the **config.csv** parameters used in the application. **NOTE:** All of the config parameters are also accessible via session attributes as explained in Chapter 5 – Session Attributes.

| PARAMETER | DESCRIPTION | EXAMPLE |
|---|---|---|
| tenant | **MANDATORY –**<br>Defines the tenant name used for your application. This is also the name used for the sub-folder in your **prompts/** folder. | sample |
| callFlow | **MANDATORY –**<br>Defines the main callflow csv file that will be used in your application. A tenant can also have multiple sub callflows. Please review Chapter 5 - BLOCK TYPES - SUB for more details. | sample-callflow.csv |
| ttsFile | **MANDATORY –**<br>Defines the TTS csv file that will be used in your application. The TTS configuration files defines all the TTS scripts used in your application. The TTS configuration file supports multiple languages. | sample-tts.csv |
| holidayFile | **OPTIONAL –**<br>Defines the Holidays configuration for your application. It lists all the holiday dates observed by your tenant application. | sample-holidays.csv |
| connectFlowType | **MANDATORY –**<br>Defines whether your application is being used for Inbound or Outbound calls. | INBOUND/OUTBOUND |
| defaultDNIS | **MANDATORY –**<br>Defines the first BLOCK ID to execute in your callflow csv file if a specific DNIS record is not found in your call flow. See Chapter 5 - BLOCK TYPES - DNIS for more details on how you can route to different starting blocks based on DNIS. | IVR100 |
| defaultQueue | **MANDATORY –**<br>Defines the default queue name used for your tenant. If no queue is needed for your application, use the default BasicQueue created with your Amazon Connect instance. | BasicQueue |
| language | **MANDATORY –**<br>Defines the default language for your application. This will usually be set to en-US. | en-US / fr-CA |

| | | |
|---|---|---|
| globalMenuAttempts | **MANDATORY –**<br>Defines the maximum number of re-attempts for each Menu or Input Block in your application. | 2 |
| globalInvalidPrompt | **MANDATORY –**<br>Defines the default prompts used for invalid menu attempts by a caller. You can define different prompts to be played based on each invalid attempt up to the maximum number of attempts defined by globalMenuAttempts. Each prompt is separated by a "\|" pipe delimiter. See Chapter 5 – BLOCK TYPES - PLAY for more details on usage. These prompts are non-bargeable. | tts:InvalidInput1\|tts:InvalidInput2 |
| globalTimeoutPrompt | **MANDATORY –**<br>Defines the default prompts used for no entry attempts by a caller. You can define different prompts to be played based on each no entry attempt up to the maximum number of attempts defined by globalMenuAttempts. Each prompt is separated by a "\|" pipe delimiter. See Chapter 5 – BLOCK TYPES - PLAY for more details on usage. These prompts are non-bargeable. | tts:NoInput1\|tts:NoInput2 |
| globalMaxPrompt | **MANDATORY –**<br>Defines the final prompt played if a caller reaches the maximum number of attempts defined by globalMenuAttempts. This is a single prompt only and is non-bargeable. | tts:MaxInputXfer |
| globalMaxAction | **MANDATORY –**<br>Defines the Block ID to execute if a caller reaches the maximum number of attempts defined by globalMenuAttempts. This usually goes to the starting block of a Pre-Queue flow or a graceful disconnect block. | PQ100 |
| globalMenuRepeat | **OPTIONAL –**<br>Defines a global DTMF key used to allow a caller to repeat any Menu within your application. This global setting can be overridden if the key is defined locally for a Menu. | * |
| globalMenuPrevious | **OPTIONAL –**<br>Defines a global DTMF key used to allow a caller to go to a previous menu. The | 9 |

| | | |
|---|---|---|
| | CompuCom RDF tracks a caller as they traverse through different menu's using a MenuTracker.<br><br>If a global setting is defined but there's a specific menu that you don't want a caller to be able to return to, you can override the previous menu key for that Menu. An example of this would be for the Language Menu. | |
| globalZeroPrompt | **OPTIONAL –**<br>Used to suppress an option Zero prompt if a caller is calling outside of business hours. If this prompt is defined within a Menu block, the prompt will only play during business hours and suppressed outside of business hours. See the sample call flow to view an example of how this is used. | agent |
| globalInputTerm | **OPTIONAL –**<br>Defines the key used to terminate variable length input from a caller using the INPUT block. | # |
| speechAnalytics | **OPTIONAL –**<br>Determines whether speechAnalytics should be turned **off** or set to **real** time or **post** time analytics. If not specified, the default value is set to **off**. | off/real/post |
| chatAnalytics | **OPTIONAL –**<br>Determines whether chatAnalytics should be turned **off** or **on**. If not specified, the default value is set to **off**. | off/on |

## <tenant>-callflow.csv

The **<tenant>-callflow.csv** file is the main callflow application file used by the RDF Base Contact Flows to read instructions on how to navigate a call flow. Using the callflow csv file replaces the need to create or update any Amazon Connect Contact flows for your application. Replace **<tenant>** with the tenant name defined in **config.csv**. You can have multiple callflow files developed as sub call flow applications. To call a sub call flow, refer to Chapter 5 – BLOCK TYPES - SUB. Refer to Chapter 5 -Application Development for additional details on how to configure your call flow blocks.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 2 | IVR100 | PLAY | wav:greeting | IVR125 |
| 3 | IVR125 | MENU | wav:language | 2:IVR125F\|9:IVR150\|*:IVR150\|timeout:IVR150\|invalid:IVR150 |
| 4 | IVR125F | LANGUGE | fr-CA | IVR150 |
| 5 | IVR150 | PLAY | eval:util.getSysAdmin("BM01") | IVR151 |
| 6 | IVR151 | PLAY | eval:util.getSysAdmin("BM02") | IVR152 |
| 7 | IVR152 | PLAY | eval:util.getSysAdmin("BM03") | IVR175 |
| 8 | IVR175 | SETQUEUE | en-US:BasicQueue\|fr-CA:BasicQueue | IVR200 |
| 9 | IVR200 | MENU | tts:mainmenu\|tts:agent | 1:IVR300\|2:IVR400\|3:IVR500\|0:PQ100 |
| 10 | IVR300 | MENU | tts:promptsmenu\|tts:agent | 1:IVR310\|2:IVR320\|3:IVR330\|4:IVR340\|0:PQ100 |
| 11 | IVR310 | INPUT | tts:getinput | var:my_variable\|length:5\|action:IVR311\|seconds:3 |
| 12 | IVR311 | PLAY | tts:ascardinal\|cardinal:session["my_variable"] | IVR312 |

## <tenant>-tts.csv

The **<tenant>-tts.csv** file is the application's TTS definition file. Replace **<tenant>** with the tenant name defined in **config.csv**. This file contains the script labels and verbiage for your application scripts used by Amazon Polly to synthesize your IVR application prompts in multiple languages. The TTS entries in this configuration file also supports SSML tags. To see a list of supported SSML tags, please refer to the following:

https://docs.aws.amazon.com/connect/latest/adminguide/supported-ssml-tags.html

**IMPORTANT:** When using SSML tags for your application scripts, DO NOT include the opening and closing **<speak>** and **</speak>** tags as the CompuCom RDF automatically inserts these tags at runtime.

| | A | B | C |
|---|---|---|---|
| 1 | LABEL | en-US | fr-CA |
| 2 | greeting | Thank you for calling the Compucom Rapid Development Framework sample application. | Merci d'avoir appelé Compucom - Rapid Development Framework, exemple d'application. |
| 3 | language | For service in English, press 1 or stay on the line. Pour obtenir des renseignements en français, faites le deux | |
| 4 | BM01 | Broadcast Message 1. You can enable and disable this message using the IVR Administration line. | Message de diffusion 1. Vous pouvez activer et désactiver ce message à l'aide de la ligne d'administration IVR. |
| 5 | BM02 | Broadcast message 2. While you test out some of the features of this sample application, you can always press star to repeat a menu, or press 9 to return to the previous menu. | Il s'agit du message de diffusion 2. Pendant que vous testez certaines des fonctionnalités de cet exemple d'application, vous pouvez toujours appuyer sur étoile pour répéter un menu ou appuyer sur 9 pour revenir au menu précédent. |

The LABEL column is used to reference the application scripts in your call flows. The different language columns contain the verbiage for each Script Label in the appropriate language. See Chapter 5 – BLOCK TYPES - PLAY for more details.

## <tenant>-holidays.csv

The **<tenant>-holidays.csv** file is the application's holidays definition file. Replace **<tenant>** with the tenant name defined in **config.csv**. This configuration file lists all the holidays observed by your tenant. Each tenant has its own holiday file. However, holiday definition files can also be centralized and shared by a common S3 bucket from CompuComRDF. The format for DATE should be in **M/Y/DDDD** format (do not include leading 0's when defining Month's or Day's).

| | A | B |
|---|---|---|
| 1 | DATE | DESCRIPTION |
| 2 | 12/25/2022 | Christmas Day |
| 3 | 12/26/2022 | Boxing Day |
| 4 | 12/27/2022 | Christmas Day |
| 5 | 1/1/2023 | New Years Day |
| 6 | 1/2/2023 | New Years Day |
| 7 | 2/20/2023 | Family Day |
| 8 | 4/7/2023 | Good Friday |
| 9 | 5/22/2023 | Victoria Day |
| 10 | 7/1/2023 | Canada Day |
| 11 | 8/7/2023 | Civic Holiday |
| 12 | 9/4/2023 | Labour Day |
| 13 | 10/9/2023 | Thanksgiving Day |
| 14 | 12/25/2023 | Christmas Day |
| 15 | 12/26/2023 | Boxing Day |
| 16 | 1/1/2024 | New Years Day |
| 17 | | |

## <tenant>-sysadmin.csv

The **<tenant>-sysadmin.csv** file is the application's System Administration file. Replace **<tenant>** with the tenant name defined in **config.csv**. This configuration file is used to define your application's Broadcast Messages and Dynamic Menus. For Broadcast Messages, refer to Chapter 5 – BLOCK TYPES - PLAY. For Dynamic Menus, refer to Chapter 5 – BLOCK TYPES – MENU. In both cases, they use the **eval** type to retrieve dynamic settings. Also review Chapter 6 - System Administration to learn how these Broadcast Messages and Dynamic Menus can be managed by the IVR System Administration Line.

| | A | B |
|---|---|---|
| 1 | KEY | VALUE |
| 2 | APPCODE | 1234 |
| 3 | VM | FALSE |
| 4 | BM01 | tts:BM01 |
| 5 | BM02 | tts:BM02 |
| 6 | BM03 | tts:BM03 |
| 7 | DM100 | 1:IVR310\|2:IVR320\|3x:IVR330\|4x:IVR340\|0:PQ100 |
| 8 | | |

# Chapter 5: Application Development

This chapter describes all the different building blocks required to develop an application with the CompuCom RDF. The foundation of the framework is driven by CSV files that can be created and modified using a simple CSV Editor such as Microsoft Excel. When saving a csv file, select the *"CSV UTF-8 (Comma delimited)(*.csv)"* option shown below:


## CALL FLOWS

The Call Flow CSV file consists of 4 columns – BLOCK, TYPE, PARAM, and ACTION shown here:

| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

### BLOCK

The BLOCK column contains unique Block ID's that identifies each of the specific call flow blocks of an application. The Block ID's are user-defined and are usually based on the Call Flow labels of a Call Flow Design Document. The Block ID is also used to capture Caller Dispositions (see Chapter 7 – Contact Dispositions). When used with a DNIS Block Type, the Block ID should contain the DNIS of the dialed phone number. By associating multiple DNIS' to the same call flow, different ACTION's (or caller treatment) can be taken based on the dialed phone number.

### TYPE

The TYPE column is used to define the function of a specific call flow block. Each Block TYPE provides different functions as outlined in section BLOCK TYPES.

### PARAM

The PARAM column is used to set parameters specific to a Block TYPE. The parameters used for each Block TYPE are outlined in section BLOCK TYPES.

### ACTION

The ACTION column is used to set the next set of actions based on the Block TYPE. The action parameters used for each Block Type are outlined in section BLOCK TYPES.

## Session Attributes

Session attributes are like Contact Attributes used in Amazon Connect Contact Flows. Unlike Contact attributes, Session attributes are used within the CompuCom RDF namespace and does not get saved within the Contact Trace Record (CTR). Session attributes can contain temporary storage to store variables within your application that you don't want to include within your Contact Trace Records or reports. To save attributes to a Contact Trace Record, use the USERDATA Block Type as described in Chapter 5 – BLOCK TYPES - USERDATA.

Session attributes can also be used in your applications for computational expressions or logical evaluations such as a loop counter, arithmetic, and conditional statements. You can store user input, save data from a backend web service, and use the contents of these attributes throughout your applications.

The Python syntax used to initialize a Session attribute is:

**session['attribute_name'] = attribute_value**

Where, **attribute_name** is a unique name for your attribute, and **attribute_value** can contain a fixed value or the name of another session attribute. The session attribute can contain any Python Built-in Data Types.

## Python Built-in Data Types

Session attributes can store data of different types, and different types can do different things. Python has the following data types built-in by default, in these categories:

| | |
|---|---|
| Text Type: | str |
| Numeric Types: | int, float, complex |
| Sequence Types: | list, tuple, range |
| Mapping Type: | dict |
| Set Types: | set, frozenset |
| Boolean Type: | bool |
| Binary Types: | bytes, bytearray, memoryview |
| None Type: | NoneType |

There is no limit to the number of Session attributes you can use in your applications.

# BLOCK TYPES

## PLAY

The PLAY block is used to play a series of application scripts in different formats. The list of formats is extensive, and the values assigned can either be fixed or a Session variable.

| TYPE | PARAM | ACTION |
|---|---|---|
| PLAY | type:value\|type:value | nextBlock |

PARAM

Defines a series of application scripts in the format **type:value** separated by the pipe "|" delimiter.

| | |
|---|---|
| type | **wav** – play a wav file from your application's S3 bucket.<br><br>**tts** – play tts scripts referenced in your <tenant>-tts.csv.<br><br>**var** – play the contents of a session variable.<br><br>**eval** – play a value returned from a Python expression or pre-built function.<br><br>**text** – play free text.<br><br>**currency** – play currency in the format of <dollarvalue>.<centsvalue> (i.e., 123.45)<br><br>**characters or spell-out –** spells out each letter of the text, as in a-b-c.<br><br>**cardinal or number –** Interprets the numerical text as a cardinal number, as in 1,234.<br><br>**slowcharacters –** inserts a small pause when spelling out each letter of the text.<br><br>**ordinal – i**nterprets the numerical text as an ordinal number, as in 1,234th.<br><br>**digits –** spells out each digit individually, as in 1-2-3-4.<br><br>**fraction –** interprets the numerical text as a fraction. This works for both common fractions such as 3/20, and mixed fractions, such as 2 ½. See below for more information.<br><br>**unit – i**nterprets a numerical text as a measurement. The value should be either a number or a fraction followed by a unit with no space in between as in 1/2inch, or by just a unit, as in 1meter. |

| | |
|---|---|
| | **date –** interprets the text as a date. The format must be yyyymmdd you can make Amazon Polly skip parts of the date using question marks.<br><br>**time –** interprets the numerical text as duration, in minutes and seconds, as in 1'21".<br><br>**address** – interprets the text as part of a street address.<br><br>**expletive** – "beeps out" the content included within the tag.<br><br>**telephone –** interprets the numerical text as a 7-digit or 10-digit telephone number, as in 2025551212. You can also use this value for handle telephone extensions, as in 2025551212x345. |
| value | A fixed value or a session attribute |

ACTION
Defines the Next BLOCK to execute in the call flow.

| | |
|---|---|
| nextBlock | The next **BLOCK ID** to execute.<br><br>**DISCONNECT** – to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## MENU

The MENU block is used to play a series of application scripts in different formats to prompt a caller to select a DTMF menu option. The caller will navigate to the next BLOCK based on the DTMF menu option selected.

| TYPE | PARAM | ACTION |
|---|---|---|
| MENU | type:value\|type:value | option:nextBlock\|param:value |

PARAM
Defines a series of application scripts in the format **type:value** separated by the pipe "|" delimiter.

| | |
|---|---|
| type | **wav** – play a wav file from your application's S3 bucket. |

| | |
|---|---|
| | **tts** – play tts scripts referenced in your <tenant>-tts.csv.<br><br>**var** – play the contents of a session variable.<br><br>**eval** – play a value returned from a Python expression or pre-built function.<br><br>**text** – play free text.<br><br>**currency** – play currency in the format of <dollarvalue>.<centsvalue> (i.e., 123.45)<br><br>**characters or spell-out –** spells out each letter of the text, as in a-b-c.<br><br>**slowcharacters –** inserts a small pause when spelling out each letter of the text.<br><br>**cardinal or number –** Interprets the numerical text as a cardinal number, as in 1,234.<br><br>**ordinal – i**nterprets the numerical text as an ordinal number, as in 1,234th.<br><br>**digits –** spells out each digit individually, as in 1-2-3-4.<br><br>**fraction –** interprets the numerical text as a fraction. This works for both common fractions such as 3/20, and mixed fractions, such as 2 ½. See below for more information.<br><br>**unit – i**nterprets a numerical text as a measurement. The value should be either a number or a fraction followed by a unit with no space in between as in 1/2inch, or by just a unit, as in 1meter.<br><br>**date –** interprets the text as a date. The format must be yyyymmdd you can make Amazon Polly skip parts of the date using question marks.<br><br>**time –** interprets the numerical text as duration, in minutes and seconds, as in 1'21".<br><br>**address** – interprets the text as part of a street address.<br><br>**expletive** – "beeps out" the content included within the tag.<br><br>**telephone –** interprets the numerical text as a 7-digit or 10-digit telephone number, as in 2025551212. You can also use this value for handle telephone extensions, as in 2025551212x345. |
| value | A fixed value or a session attribute |

ACTION

Defines a set of actions based on a caller's response in the format **option:nextBlock**. Actions are separated by the pipe "|" delimiter.

| | |
|---|---|
| option | **0-9, *, #** (optional)**–** Sets the action for a DTMF option. If a DTMF option is not defined, it will be treated as invalid input or behave based on global settings defined in config.csv (i.e., Option * for repeat or Option 9 for previous menu).<br><br>**timeout** (optional) – Sets the next action if a caller does not make a menu selection (i.e., No Input). If timeout is not defined, it will inherit the global settings of globalTimeoutPrompt, globalMaxtriesPrompt, and globalMaxtriesAction as defined in config.csv.<br><br>**invalid** (optional) **–** Sets the next action if a caller makes an invalid selection (i.e., Invalid Input). If invalid is not defined, it will inherit the global settings of globalInvalidPrompt, globalMaxtriesPrompt, and globalMaxtriesAction as defined in config.csv.<br><br>**maxaction** (optional) **–** Sets the next action if a caller reaches the maximum number of re-tries as defined by globalMenuAttempts. If maxaction is not defined, it will inherit the global settings of globalMaxtriesAction as defined in config.csv. |
| param | **barge** (optional) – Sets barge-in to **true** or **false**. When not specified, default is true. |
| nextBlock | The next BLOCK ID to execute.<br><br>**DISCONNECT** – to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## INPUT

The INPUT block is used to play a series of application scripts in different formats to prompt a caller to input a DTMF string that gets saved to a session attribute. You must specify a setting for globalInputTerm in config.csv if you expect the caller to complete their input using a terminating key.

| TYPE | PARAM | ACTION |
|---|---|---|
| INPUT | type:value\|type:value | param:value\|param:value |

### PARAM

Defines a series of application scripts in the format **type:value** separated by the pipe "|" delimiter.

| | |
|---|---|
| type | **wav** – play a wav file from your application's S3 bucket. |

| | |
|---|---|
| | **tts** – play tts scripts referenced in your <tenant>-tts.csv.<br><br>**var** – play the contents of a session variable.<br><br>**eval** – play a value returned from a Python expression or pre-built function.<br><br>**text** – play free text.<br><br>**currency** – play currency in the format of <dollarvalue>.<centsvalue> (i.e., 123.45)<br><br>**characters or spell-out –** spells out each letter of the text, as in a-b-c.<br><br>**cardinal or number –** Interprets the numerical text as a cardinal number, as in 1,234.<br><br>**slowcharacters –** inserts a small pause when spelling out each letter of the text.<br><br>**ordinal – i**nterprets the numerical text as an ordinal number, as in 1,234th.<br><br>**digits –** spells out each digit individually, as in 1-2-3-4.<br><br>**fraction –** interprets the numerical text as a fraction. This works for both common fractions such as 3/20, and mixed fractions, such as 2 ½. See below for more information.<br><br>**unit – i**nterprets a numerical text as a measurement. The value should be either a number or a fraction followed by a unit with no space in between as in 1/2inch, or by just a unit, as in 1meter.<br><br>**date –** interprets the text as a date. The format must be yyyymmdd you can make Amazon Polly skip parts of the date using question marks.<br><br>**time –** interprets the numerical text as duration, in minutes and seconds, as in 1'21".<br><br>**address** – interprets the text as part of a street address.<br><br>**expletive** – "beeps out" the content included within the tag.<br><br>**telephone –** interprets the numerical text as a 7-digit or 10-digit telephone number, as in 2025551212. You can also use this value for handle telephone extensions, as in 2025551212x345. |
| value | A fixed value or a session attribute |

ACTION

Defines a set of action parameters to define the expected DTMF INPUT. Action parameters are separated by the pipe "|" delimiter.

| | |
|---|---|
| param | **length** (mandatory)– Defines the expected length of the DTMF input string.<br><br>**action** (mandatory) **–** Defines the next Block to execute when the caller successfully enters the expected number of DTMF digits based on length.<br><br>**var** (mandatory) **–** Defines the session attribute to save the caller's DTMF input.<br><br>**barge** (optional) – Sets barge-in to **true** or **false**. When not specified, default is true.<br><br>**seconds** (optional) – Defines the number of seconds to wait between DTMF input (i.e., interdigit timeout value). If not set, default value is 10.<br><br>**timeout** (optional) – Sets the next action if a caller does not input any DTMF (i.e., No Input). If timeout is not defined, it will inherit the global settings of globalTimeoutPrompt, globalMaxtriesPrompt, and globalMaxtriesAction as defined in config.csv.<br><br>**invalid** (optional) **–** Sets the next action if a caller makes an invalid selection (i.e., Invalid Input). If invalid is not defined, it will inherit the global settings of globalInvalidPrompt, globalMaxtriesPrompt, and globalMaxtriesAction as defined in config.csv.<br><br>**maxaction** (optional) **–** Sets the next action if a caller reaches the maximum number of re-tries as defined by globalMenuAttempts. If maxaction is not defined, it will inherit the global settings of globalMaxtriesAction as defined in config.csv.<br><br>**0-9, *, #** (optional) **–** Sets the action for the first DTMF key entered by a caller. If not defined, the first digit will be included within the full DTMF string entered by a caller. **IMPORTANT:** Do not set this option if the first DTMF digit is a valid value for the remaining input. |
| value | The next **BLOCK ID** to execute.<br><br>**DISCONNECT** – to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## LANGUAGE

The LANGUAGE block is used to set the language. Once set, all prompts that follow will play in the appropriate language for both audio wav files and TTS.

| TYPE | PARAM | ACTION |
|---|---|---|
| LANGUAGE | value | nextBlock |

PARAM
Defines the ISO standard language code.

| | |
|---|---|
| value | ISO standard language code (en-US, fr-CA) |

ACTION
Defines the Next BLOCK to execute in the call flow.

| | |
|---|---|
| nextBlock | The next **BLOCK ID** to execute.<br><br>**DISCONNECT** – to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## SUB

The SUB block is used to invoke another call flow in your application. Sub call flows are reusable and repeatable sections of a call flow to create common functions. For example, you can create a Credit Card payment flow and use this as a SUB call flow reused by other applications or tenants. **IMPORTANT:** When creating a SUB call flow, the starting BLOCK ID must be set to **"START"** and the final action should be set to **"RETURN"** or **"DISCONNECT"**.

| TYPE | PARAM | ACTION |
|---|---|---|
| SUB | value | nextBlock |

PARAM
Defines the sub call flow to invoke.

| | |
|---|---|
| value | The name of the Sub callflow csv file. **NOTE:** You do not need to include the file extension. |

ACTION

Defines the Next BLOCK to execute when the Sub Call flow returns.

| | |
|---|---|
| nextBlock | The next **BLOCK ID** to execute.<br><br>**DISCONNECT** – to end the call.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## EXEC

The EXEC block provides access to a dynamically generated Python expression such as assigning a value to a session attribute, arithmetic, incrementing or decrementing a counter, or to invoke any pre-built functions. The EXEC block does not return a result. **NOTE:** To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only.

| TYPE | PARAM | ACTION |
|---|---|---|
| EXEC | expression | nextBlock |

PARAM

Used to define a Python expression to execute.

| | |
|---|---|
| expression | A valid Python expression. (i.e., session[“counter”] += 1) |

ACTION

Defines the Next BLOCK to execute.

| | |
|---|---|
| nextBlock | The next **BLOCK ID** to execute.<br><br>**DISCONNECT** – to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## EVAL

The EVAL block provides access to a dynamically generated Python expression, such as evaluating the results of a session attribute, conditional statement, or a pre-built function. The EVAL block evaluates the expression and saves the result in a **session["eval_response"]** attribute and branches based on whether the result is **true** or **false**. **NOTE:** To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only.

| TYPE | PARAM | ACTION |
|---|---|---|
| EVAL | expression | true:nextBlock\|false:nextBlock |

PARAM

Used to define a Python expression to execute.

| | |
|---|---|
| expression | A valid Python expression. (i.e., session["counter"] == 3) |

ACTION

Defines the Next BLOCK to execute.

| | |
|---|---|
| true | The next BLOCK if the evaluated expression is true. |
| false | The next BLOCK if the evaluated expression is false. |
| nextBlock | The next **BLOCK ID** to execute.<br><br>**DISCONNECT** – to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## DNIS

The DNIS block is used to provide different caller treatment based on the dialed phone number. Different ACTION's can be set to execute different BLOCK's based on DNIS. **NOTE:** the BLOCK column for DNIS should contain the dialed phone number.

| TYPE | PARAM | ACTION |
|---|---|---|
| DNIS | expression (optional) | nextBlock |

### PARAM

Like the EXEC block, the PARAM field can be used to include a dynamically generated Python expression. This is useful for initializing any session attributes you may want to initialize based on DNIS. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only.

| | |
|---|---|
| expression | A valid Python expression. (i.e., session["app"] = "custom") |

### ACTION

Defines the Next BLOCK to execute in the call flow.

| | |
|---|---|
| nextBlock | The next **BLOCK ID** to execute.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## CASE

The CASE block is used as a conditional switch statement to route callers based on the evaluation of a session attribute, Python expression, or a pre-built function. Refer to Chapter 7 – RDF Built-In Functions for a list of pre-built functions that can be used in this block type.

| TYPE | PARAM | ACTION |
|---|---|---|
| CASE | expression | result:nextBlock\|result:nextBlock |

### PARAM

Like the EVAL block, the PARAM field can be used to define a Python expression. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only.

| | |
|---|---|
| expression | A valid Python expression. (i.e., session["result"]) |

### ACTION

Defines the Next BLOCK to execute in the call flow based on the result from expression.

| | |
|---|---|
| result | The expected results as evaluated from the Python expression. For example, when calling a web service, you may want to handle a caller based on the HTTP response status codes:<br><br>200:IVR500\|400:IVR600\|404:IVR700\|else:PQ100<br><br>Call flow will take the **else** path if none of the conditions are met. |

| | |
|---|---|
| nextBlock | The next **BLOCK ID** to execute.<br><br>**DISCONNECT** – to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## USERDATA

The USERDATA block is used to save a value to the **userdata** Contact attribute. Since all session attributes belong to the CompuCom RDF namespace, session attributes are never saved to the CTR. The USERDATA block can be used to save one or more values as a Python list[] or dict[] data type. The **userdata** Contact attribute will be saved in the CTR.

| TYPE | PARAM | ACTION |
|---|---|---|
| USERDATA | expression | nextBlock |

### PARAM

Like the EVAL block, the PARAM field can be used to define a Python expression. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only.

| | |
|---|---|
| expression | A valid Python expression. (i.e., session["account_number"]) |

### ACTION

Defines the Next BLOCK to execute in the call flow.

| | |
|---|---|
| nextBlock | The next **BLOCK ID** to execute.<br><br>**DISCONNECT** – to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## SETQUEUE

The SETQUEUE block is used to set the working queue based on a caller's language selection. It is best to call this block after a language selection has been made.

| TYPE | PARAM | ACTION |
|---|---|---|
| SETQUEUE | en-US:queue\|fr-CA:queue | nextBlock |

### PARAM

The PARAM field is used to set the working queue based on language.

| | |
|---|---|
| language | **en-US** – Set the working queue if the caller's language selection is English.<br><br>**fr-CA (optional)** – Set the working queue if the caller's language selection is French. This is optional if your application only supports a single language. |
| queue | The queue name as created in Chapter 2 – Queue Set Up. By defining the queue name, CompuCom RDF can obtain the Queue ID and ARN needed by Amazon Connect. |

### ACTION

Defines the Next BLOCK to execute in the call flow.

| | |
|---|---|
| nextBlock | The next **BLOCK ID** to execute.<br><br>**DISCONNECT – to end the call.**<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## QUEUE

The QUEUE block is used to transfer a caller to Queue. By calling this block, control is transferred to the **Connect-RDF-Queue** Contact Flow.

| TYPE | PARAM | ACTION |
|---|---|---|
| QUEUE | (NOT USED) | nextBlock |

### PARAM

Not Applicable. PARAM is not required for the QUEUE Block.

### ACTION

Defines the Next BLOCK to execute in the call flow.

| | |
|---|---|
| nextBlock | The first **BLOCK ID** to execute when caller enters the Queue flow. |

## XFER

The XFER block is used to transfer a caller to an external phone number.

| TYPE | PARAM | ACTION |
|---|---|---|
| XFER | phone | nextBlock |

### PARAM

The PARAM field is used to set the external phone number to transfer to.

| | |
|---|---|
| phone | The 11-digit external phone number to transfer. You must include a preceding "1" with the 10-digit phone number. |

### ACTION

Defines the Next BLOCK to execute in the call flow.

| | |
|---|---|
| nextBlock | The next **BLOCK ID** to execute.<br><br>**DISCONNECT** – to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## REST

The REST block is used to interact with a RESTful web service. *This is in preview and is subject to change.* This section assumes that the user has a thorough understanding of REST Architecture. REST stands for *representational state transfer* and is a software architecture style that defines a pattern for client and server communications over a network. REST provides a set of constraints for software architecture to promote performance, scalability, simplicity, and reliability in the system. To interact with REST APIs, the CompuCom RDF leverages the **requests** Python library

to send HTTP requests. The CompuCom RDF supports the most common HTTP methods including GET, POST, PUT, PATCH, and DELETE.

The result returned by the REST Web Service will get saved in a **session["restResponse"]** attribute.

| TYPE | PARAM | ACTION |
|---|---|---|
| REST | api:url\|operation:type\|payload:data\|secret:id | true:nextBlock\|false:nextBlock |

PARAM

The PARAM field is used to set the REST API parameters.

| | |
|---|---|
| api | **url** – defines the API URL |
| operation | **type –** defines the REST API operation (**get, post, put, patch, delete**) |
| payload (optional) | **data –** defines the input data in json format. This is required only for **post, put,** and **patch** operations. |
| secret (optional) | **id –** the secret key used to look up API keys from AWS Secrets Manager for web services that require authentication. |

ACTION

Defines the Next BLOCK to execute in the call flow.

| | |
|---|---|
| true | The next **BLOCK ID** if the REST web service is successful. |
| false | The next **BLOCK ID** if the REST web service fails. |
| nextBlock | The next **BLOCK ID** to execute.<br><br>**DISCONNECT –** to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## SOAP

The SOAP block is used to interact with a SOAP web service. *This is in preview and is subject to change.* This section assumes that the user has a thorough understanding of SOAP. The CompuCom RDF integrates with the Zeep Python Soap Client. Zeep inspects the WSDL document and generates the corresponding code to use the services and types in the document. This provides an easy-to-use programmatic interface to a SOAP server that the CompuCom RDF leverages. WSDL documents provide a number of operations (functions) per binding. A binding is a collection of operations which are called via a specific protocol. These protocols are generally Soap 1.1 or Soap 1.2.

The CompuCom RDF further simplifies the interaction with a SOAP web service by allowing a developer to specify only the WSDL URL and the operation of the protocol being used. One of the first things you will want to do is get an overview of all available operations and their call signatures by inspecting the WSDL document. There are many tools available to inspect and validate a WSDL document, such as SoapUI.

As an example, when we inspect the available operations for a simple Calculator SOAP Web Service:

http://www.dneonline.com/calculator.asmx?WSDL

The available operations and respective protocols that are found are:

```
Service: Calculator
     Port: CalculatorSoap (Soap11Binding: {http://tempuri.org/}CalculatorSoap)
         Operations:
            Add(intA: xsd:int, intB: xsd:int) -> AddResult: xsd:int
            Divide(intA: xsd:int, intB: xsd:int) -> DivideResult: xsd:int
            Multiply(intA: xsd:int, intB: xsd:int) -> MultiplyResult: xsd:int
            Subtract(intA: xsd:int, intB: xsd:int) -> SubtractResult: xsd:int

     Port: CalculatorSoap12 (Soap12Binding: {http://tempuri.org/}CalculatorSoap12)
         Operations:
            Add(intA: xsd:int, intB: xsd:int) -> AddResult: xsd:int
            Divide(intA: xsd:int, intB: xsd:int) -> DivideResult: xsd:int
            Multiply(intA: xsd:int, intB: xsd:int) -> MultiplyResult: xsd:int
            Subtract(intA: xsd:int, intB: xsd:int) -> SubtractResult: xsd:int
```

As shown above, the operations and it's respective protocols for Soap 1.1 and Soap 1.2 bindings are listed. To call the **Multiply(int, int)** operation, simply provide the WSDL URL and operation as shown below:

| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 5 | WS300 | SOAP | wsdl:http://www.dneonline.com/calculator.asmx?WSDL\|operation:Multiply(25,3) | true:WS350\|false:WS375 |
| 6 | WS350 | PLAY | text:The calculator web service returned an answer of\|characters:session['soapResponse'] | WS400 |
| 7 | WS375 | PLAY | text: The calculator web service returned an error. Sorry about that. | WS400 |

The result returned by the SOAP Web Service will get saved in a **session["soapResponse"]** attribute.

| TYPE | PARAM | ACTION |
|---|---|---|
| SOAP | wsdl:url\|operation:name\|port:name\|secret:id | true:nextBlock\|false:nextBlock |

PARAM

The PARAM field is used to set the parameters to the SOAP WSDL.

| | |
|---|---|
| wsdl | **url –** Defines the WSDL URL. |
| operation | **name** – Enter the operation to invoke as defined by the WSDL with appropriate parameters. |
| port (optional) | **name –** Specify the port binding to use (**soap1.1** or **soap1.2**). If not provided, it will use binding for **soap1.2** |
| secret (optional) | **id –** the secret key used to look up security credentials of the web service from AWS Secrets Manager. |

ACTION

Defines the Next BLOCK to execute.

| | |
|---|---|
| true | The next **BLOCK ID** if the SOAP web service is successful. |
| false | The next **BLOCK ID** if the SOAP web service fails. |
| nextBlock | The next **BLOCK ID** to execute.<br><br>**DISCONNECT –** to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

## RECORD

The RECORD block is used to play a series of application scripts in different formats to prompt a caller to record a message and save the recording to an S3 Bucket. *This is in preview and is subject to change*. The RECORD block requires an additional lambda function to be installed locally on the tenant AWS account. Information for this lambda function will be made available in a later version.

| TYPE | PARAM | ACTION |
|---|---|---|
| RECORD | type:value\|type:value | nextBlock |

PARAM

Defines a series of application scripts in the format **type:value** separated by the pipe "|" delimiter.

| | |
|---|---|
| type | **wav** – play a wav file from your application's S3 bucket.<br><br>**tts** – play tts scripts referenced in your <tenant>-tts.csv. |

| | |
|---|---|
| | **var** – play the contents of a session variable.<br><br>**eval** – play a value returned from a Python expression or pre-built function.<br><br>**text** – play free text.<br><br>**currency** – play currency in the format of <dollarvalue>.<centsvalue> (i.e., 123.45)<br><br>**characters or spell-out –** spells out each letter of the text, as in a-b-c.<br><br>**slowcharacters –** inserts a small pause when spelling out each letter of the text.<br><br>**cardinal or number –** Interprets the numerical text as a cardinal number, as in 1,234.<br><br>**ordinal – i**nterprets the numerical text as an ordinal number, as in 1,234th.<br><br>**digits –** spells out each digit individually, as in 1-2-3-4.<br><br>**fraction –** interprets the numerical text as a fraction. This works for both common fractions such as 3/20, and mixed fractions, such as 2 ½. See below for more information.<br><br>**unit – i**nterprets a numerical text as a measurement. The value should be either a number or a fraction followed by a unit with no space in between as in 1/2inch, or by just a unit, as in 1meter.<br><br>**date –** interprets the text as a date. The format must be yyyymmdd you can make Amazon Polly skip parts of the date using question marks.<br><br>**time –** interprets the numerical text as duration, in minutes and seconds, as in 1'21".<br><br>**address** – interprets the text as part of a street address.<br><br>**expletive** – "beeps out" the content included within the tag.<br><br>**telephone –** interprets the numerical text as a 7-digit or 10-digit telephone number, as in 2025551212. You can also use this value for handle telephone extensions, as in 2025551212x345. |
| value | A fixed value or a session attribute |

ACTION

Defines the Next BLOCK to execute in the call flow.

| | |
|---|---|
| nextBlock | The next **BLOCK ID** to execute. |

| | |
|---|---|
| | **DISCONNECT –** to end the call.<br><br>**RETURN** – used inside a SUB call flow.<br><br>**eval** – a dynamically generated action from a Python expression or pre-built function. To minimize any security implications, Python expressions are limited to the session namespace and RDF built-in functions only. |

# Chapter 6: System Administration

The CompuCom RDF contains a pre-built IVR System Administration module that can be re-used and "plugged-in" to other tenant applications to allow administrators to enable and disable Broadcast Messages and Dynamic Menu options. From the callflows downloaded in Chapter 2 - Prerequisites, a **sysadmin-callflow.csv** module is included that interacts with the **<tenant>-sysadmin.csv** to manage these settings. The <tenant>-sysadmin.csv file can be used to add other configurable parameters for your application.

The following is a sample **<tenant>-sysadmin.csv** file:

| | A | B |
|---|---|---|
| 1 | KEY | VALUE |
| 2 | APPCODE | 1234 |
| 3 | VM | FALSE |
| 4 | BM01 | tts:BM01 |
| 5 | BM02 | tts:BM02 |
| 6 | BM03 | tts:BM03 |
| 7 | DM100 | 1:IVR310\|2:IVR320\|3x:IVR330\|4x:IVR340\|0:PQ100 |

*__APPCODE__ is used as a pass code for administrators to access the functions available in the IVR Administration line.

## Broadcast Messages

Broadcast Messages can be used in a PLAY block with the **eval** type to call the **util.getSysAdmin()** pre-built function. The following are example broadcast messages set in **sample-callflow.csv**:

| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 5 | IVR150 | PLAY | eval:util.getSysAdmin("BM01") | IVR151 |
| 6 | IVR151 | PLAY | eval:util.getSysAdmin("BM02") | IVR152 |
| 7 | IVR152 | PLAY | eval:util.getSysAdmin("BM03") | IVR175 |

The TTS or WAV label for your broadcast message scripts should match the Broadcast Message ID (or LABEL). For example, Broadcast Message BM01 should have a matching TTS labeled BM01, or a wav file named BM01.wav. The following are TTS labels from **<tenant>-tts.csv**:

| | A | B | C |
|---|---|---|---|
| 1 | LABEL | en-US | fr-CA |
| 4 | BM01 | Broadcast Message 1. You can enable and disable broadcast messages using the IVR Administration line. | Message de diffusion 1. Vous pouvez activer et désactiver ce message à l'aide de la ligne d'administration IVR. |
| 5 | BM02 | Broadcast message 2. While you test out some of the features of this sample application, you can always press star to repeat a menu, or press 9 to return to the previous menu. | Il s'agit du message de diffusion 2. Pendant que vous testez certaines des fonctionnalités de cet exemple d'application, vous pouvez toujours appuyer sur étoile pour répéter un menu ou appuyer sur 9 pour revenir au menu précédent. |
| 6 | BM03 | Broadcast Message 3. You should also know that the default invalid and timeout retries is set to 2. | This is Broadcast Message 3. You should also know that the default invalid and timeout retries is set to 2. |

# Dynamic Menus

Dynamic Menus can be used in a MENU block with the **eval** type to call the **util.getDynamicPrompts()** and **util.getDynamicMenu()** pre-built functions. The following is an example dynamic menu set in **sample-callflow.csv**:

| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 10 | IVR300 | MENU | eval:util.getDynamicPrompt("tts","DM100") | eval:util.getDynamicMenu("DM100") |

**util.getDynamicPrompts(type, id)** – where **type** is the application script type to play (see Chapter 5 – BLOCK TYPES - PLAY), and **id** is the Dynamic Menu ID to reference in <tenant>-sysAdmin.csv.

**util.getDynamicMenu(id)** – where **id** is the Dynamic Menu ID to reference in <tenant>-sysAdmin.csv.

The TTS or WAV labels for your Dynamic Menu should have an introductory script plus a corresponding script for each dynamic menu option. For example, Dynamic Menu DM100 with 5 dynamic options (1-4, and 0), should have matching TTS entries labeled DM100 (introductory menu script), DM100-1 (script for option 1), DM100-2 (script for option 2), DM100-3 (script for option 3), DM100-4 (script for option 4), and DM100-0 (script for option 0). The **util.getDynamicPrompts()** pre-built function will always play the introductory script and only the scripts where the corresponding option is enabled.

| | A | B | C |
|---|---|---|---|
| 1 | LABEL | en-US | fr-CA |
| 8 | DM100 | This is a Dynamic Menu. You can enable and disable menu options using the IVR Administration Line | Il s'agit d'un menu dynamique. Vous pouvez activer et désactiver les options de menu à l'aide de la ligne d'administration IVR. |
| 9 | DM100-1 | To play a numerical value in different formats, press 1 | Pour lire une valeur numérique dans différents formats, appuyez sur 1 |
| 10 | DM100-2 | To play a date, press 2 | Pour écouter une date, appuyez sur 2 |
| 11 | DM100-3 | To play a telephone number, press 3 | Pour écouter un numéro de téléphone, appuyez sur 3 |
| 12 | DM100-4 | To play an address, press 4 | Pour lire une adresse, appuyez sur 4 |
| 13 | DM100-0 | To speak with a customer service representative, press 0 | Pour parler à un représentant du service client, appuyez sur 0 |

A Dynamic Menu with all options enabled in **sample-sysAdmin.csv**:

| | A | B |
|---|---|---|
| 1 | KEY | VALUE |
| 7 | DM100 | 1:IVR310\|2:IVR320\|3:IVR330\|4:IVR340\|0:PQ100 |

A Dynamic Menu option is disabled by simply marking the option with an 'x' as shown below. This can be accomplished manually or via the IVR System Administration module

| | A | B |
|---|---|---|
| 1 | KEY | VALUE |
| 7 | DM100 | 1:IVR310\|2:IVR320\|3x:IVR330\|4x:IVR340\|0:PQ100 |

# Chapter 7: RDF Built-In Functions

## Check Business Hours

Business Hours are checked using Amazon Connect's **describe_hours_of_operation** API. A pre-built function can be used to check business hours. This function will first check to see if the current day is a holiday by reading **<tenant>-holidays.csv**, and then checks the Business Hours configuration for the working queue as set in **SETQUEUE**. The result will return **true** if the Contact Centre queue is open, and **false** if the current day is a holiday or the current time is outside of business hours.

**queues.checkBusinessHours():** CHECK HOLIDAY AND BUSINESS HOURS

## Check Holiday

While a holiday is already checked by using the **queues.checkBusinessHours()** pre-built function, the Check Holiday function is useful if a specific application script is needed to play back a holiday announcement that is different from a standard closed message. The result will return **true** if the current day is a holiday, and **false** if the current day is not a holiday.

**queues.checkHoliday():** CHECK HOLIDAY

## Queue Metrics

The CompuCom RDF leverages Amazon Connect's **get_current_metric_data()** API to obtain Queue metrics such as AGENTS_ONLINE, AGENTS_STAFFED, AGENTS_AVAILABLE, SLOTS_AVAILABLE, CONTACTS_IN_QUEUE, and OLDEST_CONTACT_AGE. This allows the CompuCom RDF to implement custom routing logic and calculations using these queue metrics. Review the following for more details:

https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/connect/client/get_current_metric_data.html

To overcome Amazon Connect's API throttling limits, exponential back off and jitter was implemented to retry failed API calls when there are too many concurrent requests. Review the following for more details:

https://aws.amazon.com/blogs/architecture/exponential-backoff-and-jitter/

The following are pre-built functions that can be called within an EVAL Block:

**queues.agentsOnline():** CHECK AGENTS ONLINE

**queues.agentsStaffed():** CHECK AGENTS STAFFED

**queues.agentsAvailable():** CHECK AGENTS AVAILABLE

**queues.slotsAvailable():** CHECK SLOTS AVAILABLE

**queues.withinQueueCapacity(threshold):** CHECK QUEUE CAPACITY

**queues.withinWaitCapacity(threshold):** CHECK WAIT THRESHOLD

Alternatively, you can make one function call:

**queues.getQueueMetrics(queueID):** GET QUEUE METRICS

When calling this one function, the following session attributes become available to the application:

**session['queuemetrics'][ 'agentsOnline']**

**session['queuemetrics']['agentsStaffed']**

**session['queuemetrics']['agentsAvailable']**

**session['queuemetrics']['slotsAvailable']**

**session['queuemetrics']['contactsInQueue']**

**session['queuemetrics']['OldestContactAge']**

## DynamoDB

The CompuCom RDF can replace CSV configuration files with DynamoDB tables. However, updating and maintaining DB tables adds a layer of complexity when importing and exporting data. Therefore, CSV files were used to implement all configuration settings. To utilize a DynamoDB table, follow the instructions in Chapter 2 – Create Cross-Account IAM Role to ensure the Cross-Account IAM role of your tenant contains the proper permissions to access DynamoDB tables.

From CompuCom RDF, you can call the **readDB(tableName, Key, Type)** built-in function to read the contents of a simple DynamoDB table with Key/Value pairs, where **tableName** is the DynamoDB table name, **Key** is the search key in the **table**, and **Type** defines the data type of the value stored in the table (i.e. “S” for String or “BOOL” for Boolean). Review the following for more information on DynamoDB Data Types:

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes

## Clear Previous Menu

The CompuCom RDF tracks all the Menus that a caller accesses within a MenuTracker as they traverse through the IVR call flow. A **util.clearPreviousMenu(session)** built-in function can be used in your call flow design that requires a particular menu to be removed from the MenuTracker. For example, a confirmation Menu might not be appropriate in some cases to return. In this case, the **util.clearPreviousMenu(session)** built-in function will remove the most recent Menu from the MenuTracker List. It's also important to note that the MenuTracker will not add a Menu that already exists in the MenuTracker List. This is to avoid call flow designs where callers are explicitly sent back to a Menu that has already been accessed.

## Contact Dispositions

The CompuCom RDF tracks a caller's disposition as they traverse through a call flow using the Block ID's created in your application. It will also track invalid and timeout outcomes. A **contactDisposition** contact attribute is used in Amazon Connect to save the caller dispositions within the CTR. An example is shown below:

| | |
|---|---|
| contactDisposition | IVR100,IVR125,IVR150,IVR151,IVR152,IVR175,IVR200,IVR200-invalid,IVR300,IVR300-timeout,IVR300-timeout,IVR300-MaxTries,PQ100,PQ110,PQ175,PQ300,PQ500,PQ700,PQ600,Q1000 |

You can also create your own custom caller dispositions to get added to the **contactDispostion** contact attribute by calling the following pre-built function with an EXEC block:

**session['contactDisposition'] = util.addDisposition(session['contactDisposition'], new_dispostion)**

Where, new_disposition is the new caller disposition to add to the **contactDisposition** contact attribute.

# Chapter 8: Helpful Hints

This chapter provides helpful hints on some of the underlying components of the CompuCom RDF and additional configuration options.

## RDF Lambda Cache

The CompuCom RDF uses temporary storage to save application configuration files during AWS Lambda runtime to avoid reading files from S3 at every invocation. Each Lambda execution environment provides between 512 MB and 10,240 MB, in 1-MB increments of disk space in the /tmp directory. The /tmp directory content is preserved for the lifetime of the execution environment, providing a transient cache that can be used for multiple invocations to optimize performance. The CompuCom RDF checks if the cache has the data required, otherwise it will fetch the required application configuration files from S3 and saves them within the /tmp directory for subsequent invocations. The /tmp directory is equivalent to a local hard disk, providing fast throughput. Operations like makedirs, listdir, etc. on this folder using the 'os' Python package can be performed just like you'd perform on a local hard disk.

At the start of every call, the CompuCom RDF will check to see if an updated configuration file exists in S3. If a new version of an application configuration file exists, the /tmp cache directory will be updated with the new configuration file. Otherwise, it will continue to use the configuration file saved in cache.

It is important to note that the /tmp directory is private to each Lambda instance/execution context and it's not shared among all Lambda instances. That means if a request is handled by a different execution context, the data won't be present there and will need to be fetched in S3.

**queues.csv**

A **queues.csv** file is automatically created into the /tmp directory by CompuCom RDF that defines all the Queue Names and Queue ID's used in your application. This configuration file allows a developer to reference Queues by Name within their applications without having to set the Queue ID that is required by Amazon Connect. Refer to Chapter 5 – BLOCK TYPES – SETQUEUE for additional details.

| | A | B |
|---|---|---|
| 1 | KEY | VALUE |
| 2 | BasicQueue | d9377d03-2e41-494b-923b-e321d94f25ea |
| 3 | | |

**Business Hours and Queue Descriptions**

Static API data returned for business hours and queue descriptions are also saved in the lambda cache to significantly reduce the number of calls to Connect API's and further circumventing any Connect API throttling limits.

## Menu Tracker

The CompuCom RDF tracks all Menu's that a caller accesses within a MenuTracker[] List. This List is used to provide the ability for a caller to return to a previous menu as they traverse through a call flow. All MENU blocks added in your application gets added to the MenuTracker only if a DTMF key is defined for the globalMenuPrevious configuration entry in config.csv. The MenuTracker alleviates the need for a developer to continually define the previous menu option within their call flow application. As you add MENU's to your application, you do not need to specify the previous menu option as the CompuCom RDF will handle this automatically. You can also override the global previous menu setting by explicitly defining the previous menu DTMF option at any given MENU. When a previous menu option is defined in a MENU, it will not get added to the MenuTracker List. You can also use the util.clearPreviousMenu(session) built-in function to remove the most recent MENU from the MenuTracker List.

Previous Menu's can also span across different sub call flows. However, when a call returns from a SUB call flow, the MenuTracker will clear all MENU's that were accessed from within that SUB call flow.

## Queue Tracker

In addition to the MenuTracker, a QueueTracker is also used to maintain the queue that was most recently set with SETQUEUE at a MENU. In most cases, a queue would have been set in the call flow when arriving at a MENU. If not set, the DefaultQueue will be used. When a caller traverses through several different menus, the QueueTracker will ensure that the correct queue will be maintained even as a caller returns to previous menus. The QueueTracker works in tandem with the MenuTracker and is only active if the globalMenuPrevious configuration entry is defined in config.csv.

## Contact Attributes

The CompuCom RDF uses limited Contact Attributes in Amazon Connect. The CTR will only include the following:

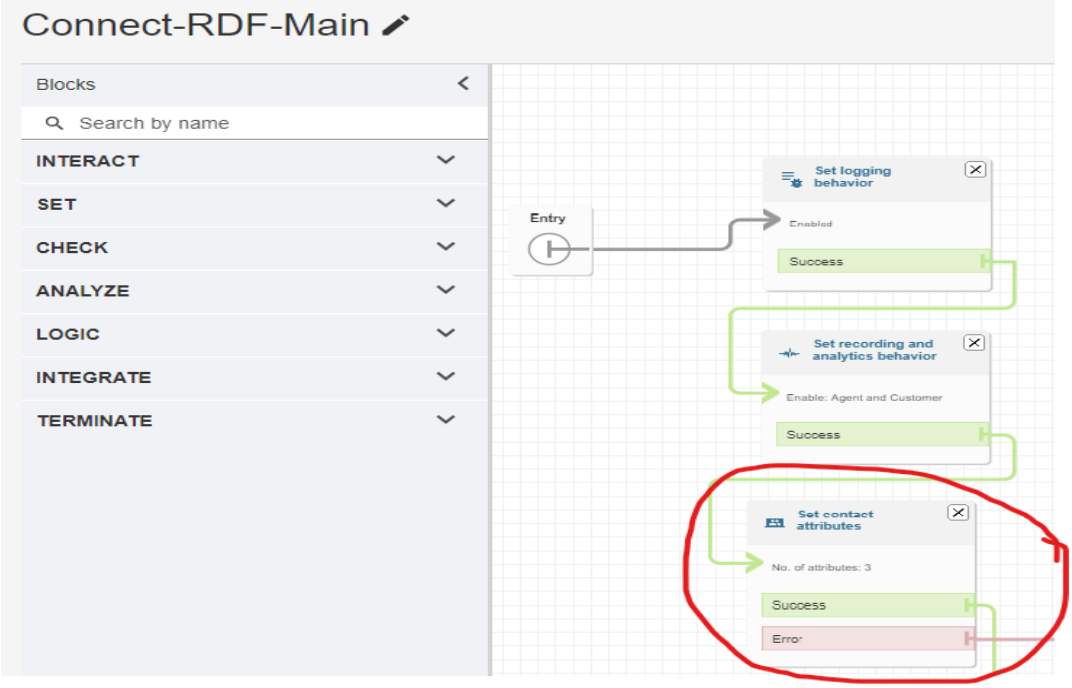
Attributes

| | |
|---|---|
| CPA | INBOUND |
| RDF | arn:aws:lambda:ca-central-1:037338416846:function:RDF:prod |
| agentTip | English |
| callFlowID | IVR1505 |
| contactDisposition | 14378805840,IVR1000,IVR1005,IVR1500Q,IVR1500A,IVR1500,IVR1505 |
| errorMessage | NA |

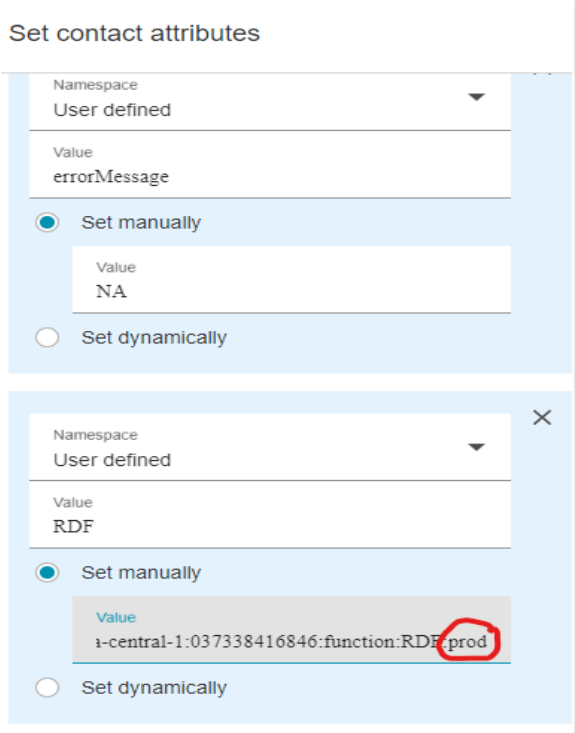
***userdata** Contact Attribute will also appear when using the USERDATA block type.

# Pointing to DEV/UAT/PROD

The CompuCom RDF can be pointed to different instances for DEV, UAT and PROD by updating the third block of the **Connect-RDF-Main** and **Connect-RDF-Agent** Contact Flows:

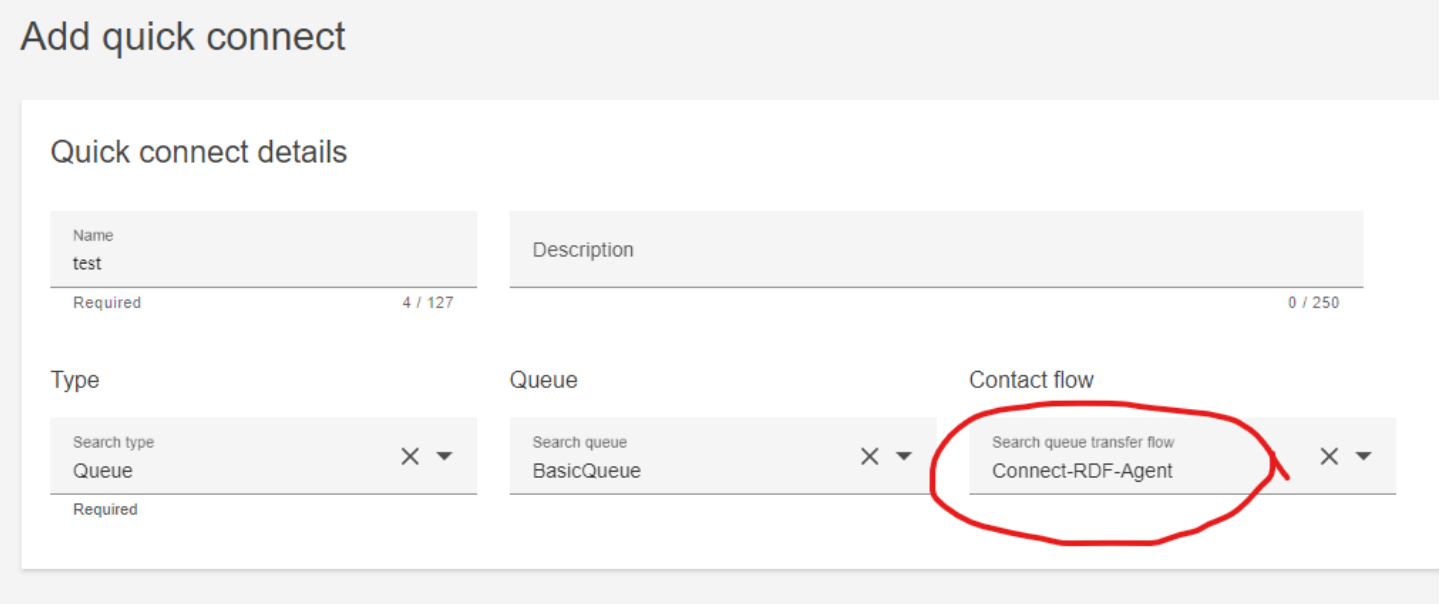



Open the Set contact attributes block in **Connect-RDF-Main** and **Connect-RDF-Agent**, and update the RDF attribute to point to either **dev, uat, or prod**, as shown below:



# Quick Connects

Quick Connects can be used to allow agents to transfer callers to another queue. When creating a Quick Connect, select **Connect-RDF-Agent** as the Contact flow:

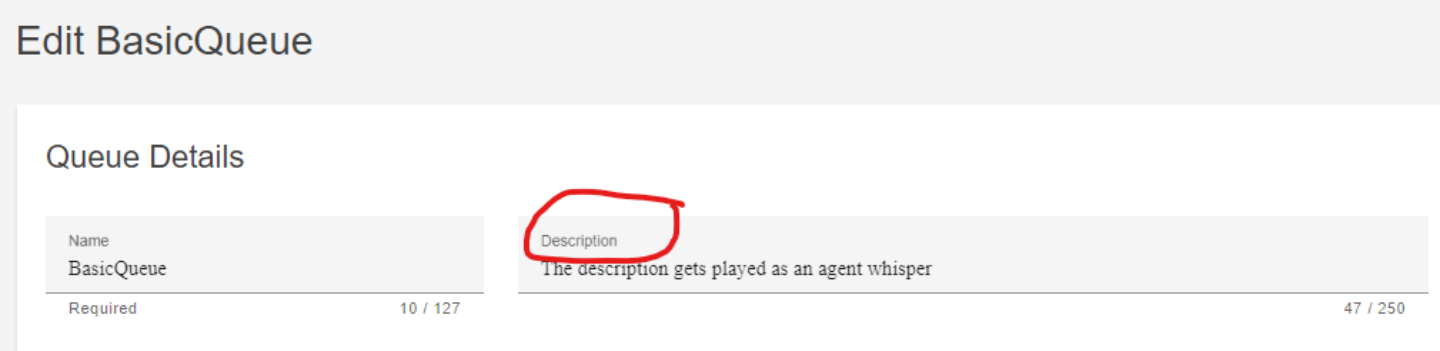



When a Quick Connect is answered by the CompuCom RDF, the **Connect-RDF-Agent** Contact Flow takes control of the application. The first block that a Quick Connect call will execute is the Block ID defined for **"globalMaxtriesAction"** in config.csv. The globalMaxtriesAction block ID is used since it defines the beginning of the pre-queue call flow. To determine if your application was reached by a Quick Connect, check for **session['function'] == 'agent'**. If this condition is true, you can design your application to interact with a Quick Connect contact. An example is shown below that allows an agent to send a caller back to the Pre-Queue flow:

| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 25 | PQ100 | EXEC | session['queuemetrics'] = queues.getQueueMetrics(session['q | PQ110 |
| 26 | PQ110 | EVAL | session['function']=='agent' | true:PQ125\|false:PQ175 |
| 27 | PQ125 | EVAL | session['queuemetrics']['agentsOnline'] > 0 | true:PQ150\|false:PQ175 |
| 28 | PQ150 | PLAY | text:The queue is open and staffed. Please transfer the caller r | PQ175 |

# Agent Whispers

The CompuCom RDF uses the **Connect-RDF-Whisper** Contact Flow to whisper the queue description to an agent. You can modify what the agent hears by updating the Queue Description from the Amazon Connect console:




The Queue Description gets saved into an **agentTip** Contact Attribute that is saved in the CTR. **NOTE**: The Queue Description is cached, so updating the Description will only take effect once a Lambda execution environment is torn down.

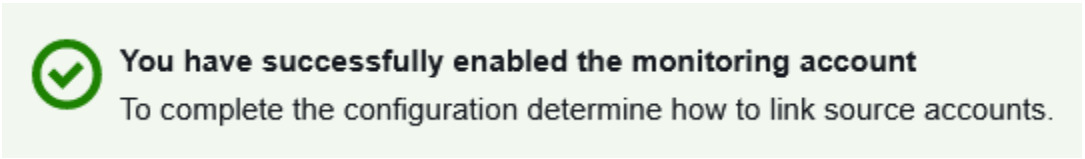
For additional information, https://docs.aws.amazon.com/lambda/latest/operatorguide/execution-environment.html

## CloudWatch Monitoring

CloudWatch Metrics & Logs from CompuCom RDF can be monitored by a tenant AWS account. Reviewing CloudWatch logs is helpful when developing and troubleshooting IVR applications with the CompuCom RDF. It provides details of a caller navigating within your IVR application and provides logs of any errors or exceptions. Currently, there is a set service limit for the number of CloudWatch Monitoring accounts that can be configured on the CompuCom RDF. The default service quota limit for monitoring accounts can be increased by creating a support case with AWS Support. For this reason, it is recommended to have monitoring accounts enabled for development instances only.

To request a tenant AWS account with CloudWatch monitoring of the CompuCom RDF, perform the following steps to configure your tenant AWS account to monitor the CompuCom RDF as a source account. Once completed, send a request to CompuCom with the **URL** information as described below:

- ➔ Go to **CloudWatch** of the Tenant AWS Account
- ➔ Go to **Settings** -> Under **"Monitoring Account Configuration"**, click **Configure** -> Under **"List source accounts"**, input the RDF Account Number -> Under **"Define a label to help identify your source account"**, click on **Account name** -> Click on **Configure**
- ➔ You should see the following message:

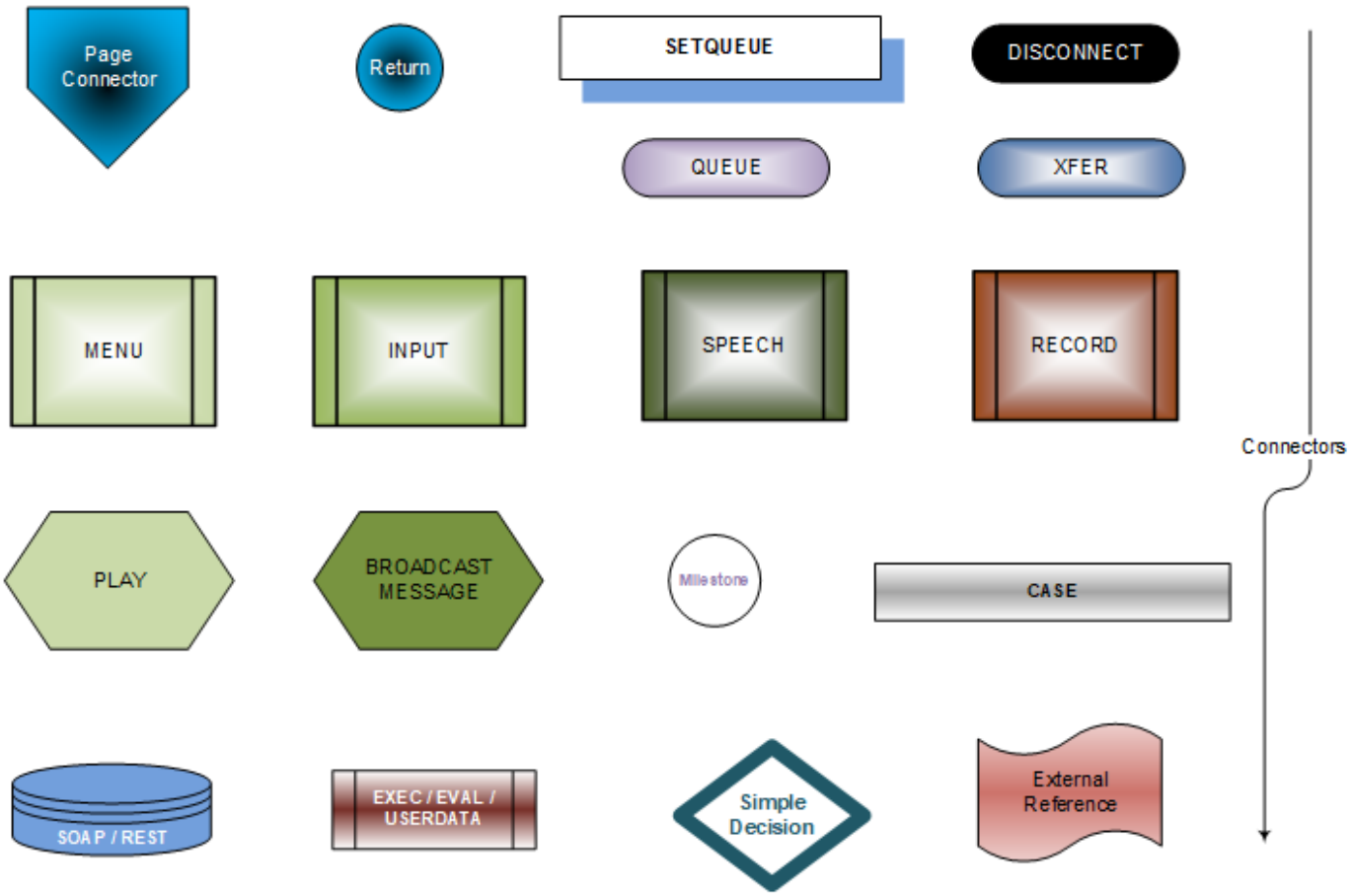


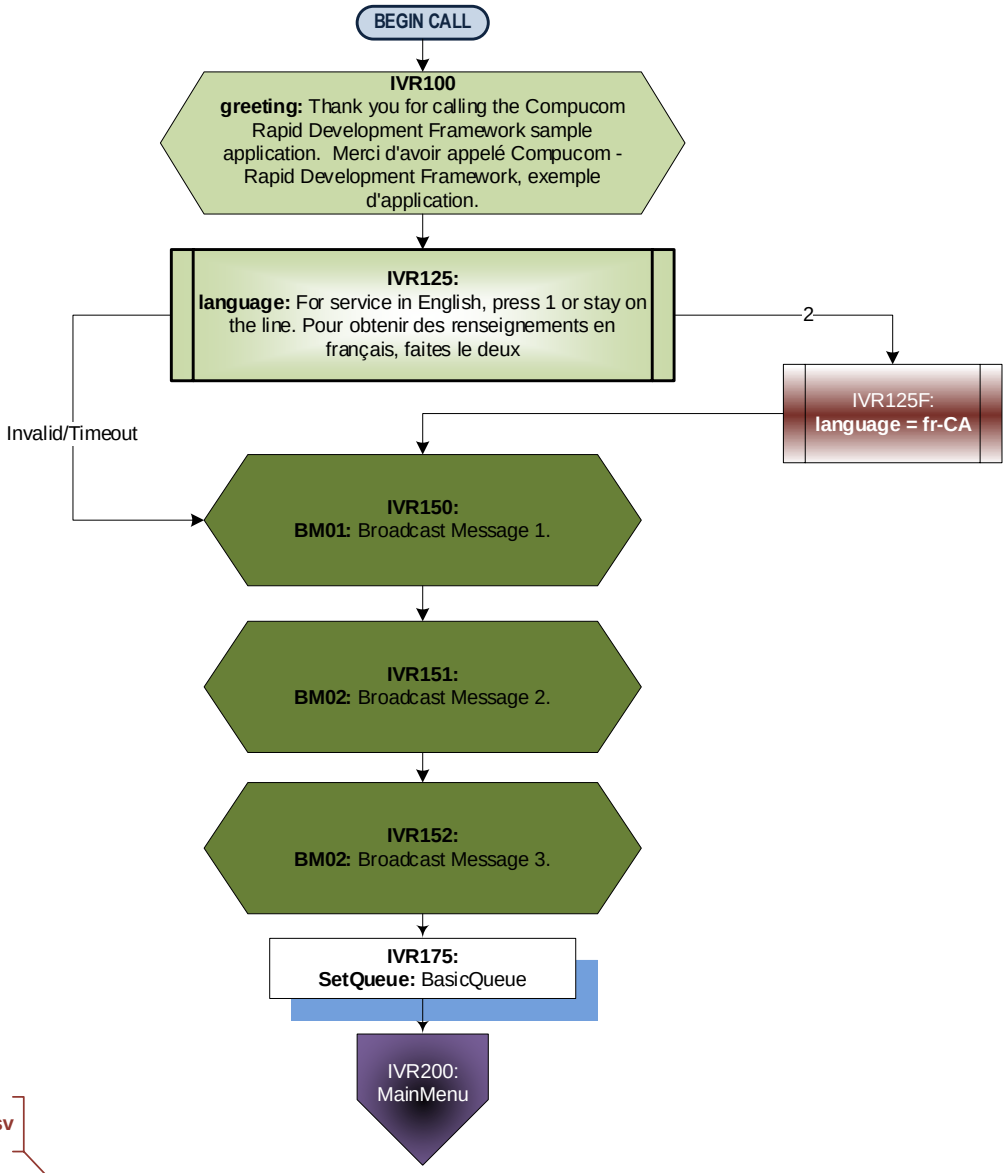

- ➔ Go back to **Settings** -> Under **"Monitoring Account Configuration"**, click **Resources to link accounts** -> Under **"Determine how to link your source accounts"** click on **Any Account** -> Under **"Any Account"**, click on **Copy URL** (this URL contains the *monitoring account sink ARN).*
- ➔ Send the URL copied from your clipboard to CompuCom.

# Chapter 9: Sample Application

This chapter provides a walkthrough of the sample application that was deployed in Chapter 3 – Hello World! For additional clarity, the call flow diagrams also include it's corresponding configuration entries in ***sample-callflow.csv.***

The shapes used in the call flow diagrams are as follows:






**sample-callflow.csv**

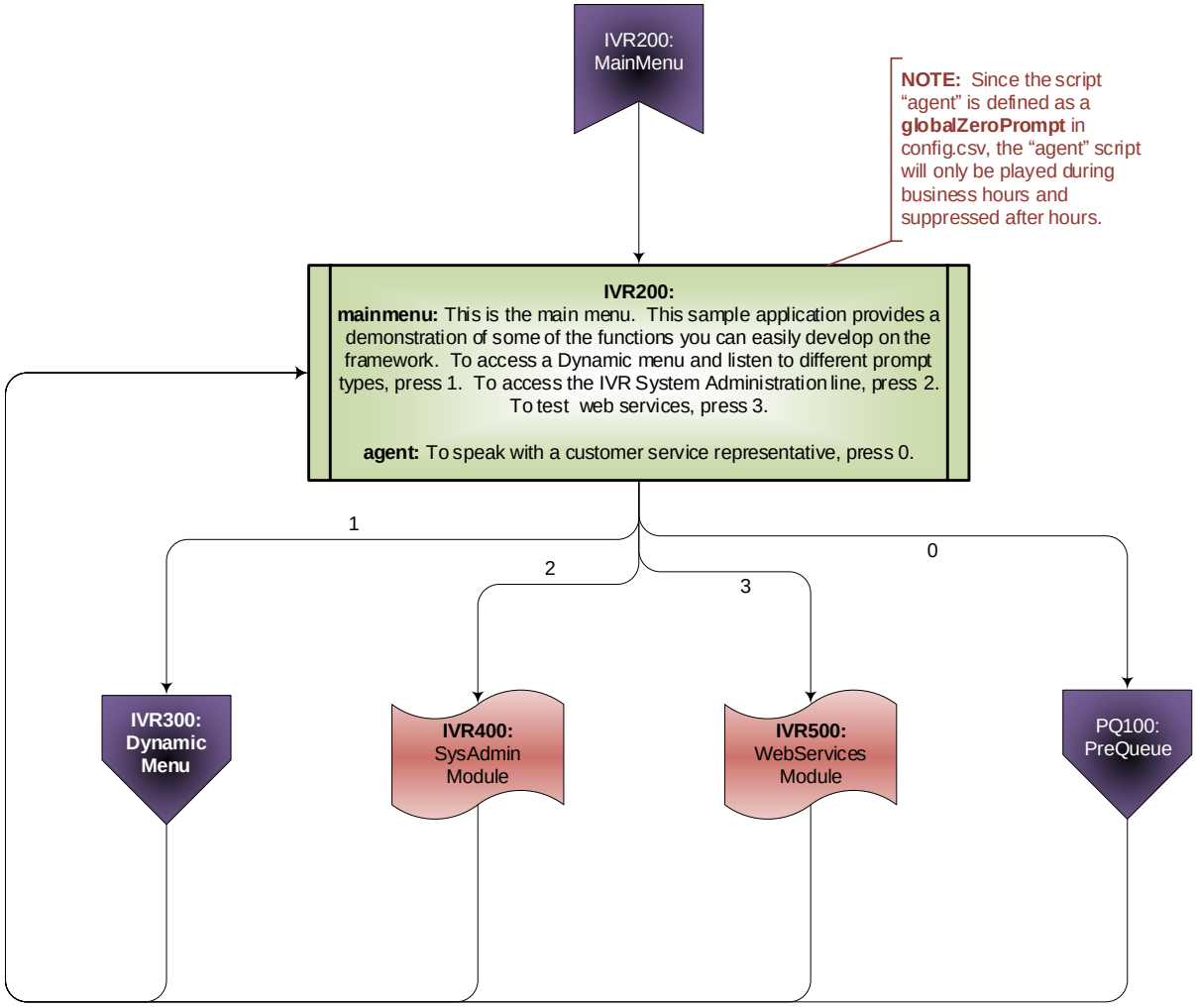
| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 2 | IVR100 | PLAY | wav:greeting | IVR125 |
| 3 | IVR125 | MENU | wav:language | 2:IVR125F\|9:IVR150\|*:IVR150\|timeout:IVR150\|invalid:IVR150 |
| 4 | IVR125F | LANGUAGE | fr-CA | IVR150 |
| 5 | IVR150 | PLAY | eval:util.getSysAdmin("BM01") | IVR151 |
| 6 | IVR151 | PLAY | eval:util.getSysAdmin("BM02") | IVR152 |
| 7 | IVR152 | PLAY | eval:util.getSysAdmin("BM03") | IVR175 |
| 8 | IVR175 | SETQUEUE | en-US:BasicQueue\|fr-CA:BasicQueue | IVR200 |

**sample-sysadmin.csv** with all broadcast messages enabled

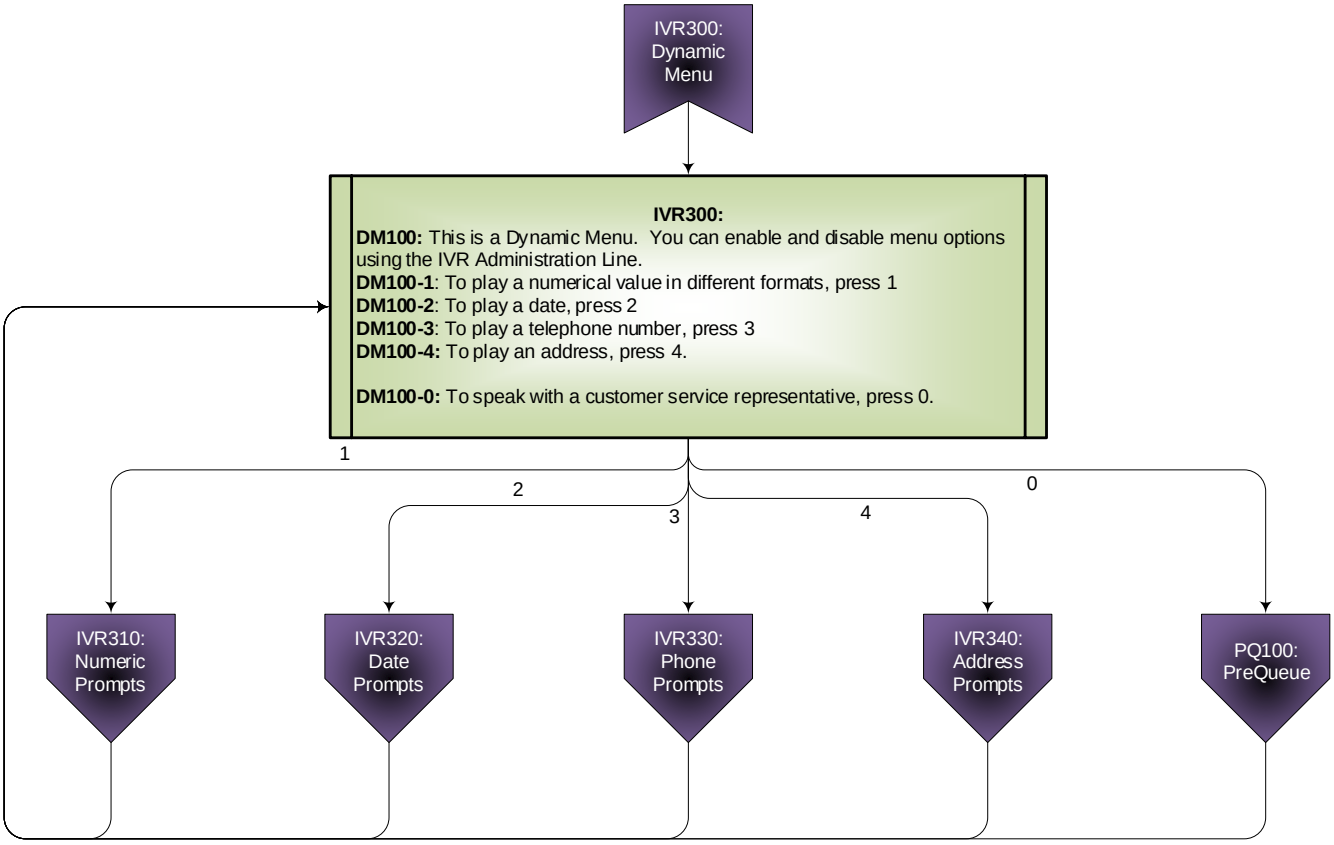
| | A | B |
|---|---|---|
| 1 | KEY | VALUE |
| 4 | BM01 | tts:BM01 |
| 5 | BM02 | tts:BM02 |
| 6 | BM03 | tts:BM03 |

**sample-sysadmin.csv** with all broadcast messages disabled

| | A | B |
|---|---|---|
| 1 | KEY | VALUE |
| 4 | BM01 | na |
| 5 | BM02 | na |
| 6 | BM03 | na |




| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 54 | IVR200 | MENU | tts:mainmenu\|tts:agent | 1:IVR300\|2:IVR400\|3:IVR500\|0:PQ100 |
| 55 | IVR300 | MENU | eval:util.getDynamicPrompt("tts","DM100") | eval:util.getDynamicMenu("DM100") |
| 56 | IVR400 | SUB | sysadmin-callflow | IVR200 |
| 57 | IVR500 | SUB | ws-callflow | IVR200 |
| 58 | PQ100 | EXEC | session['queuemetrics'] = queues.getQueueMetrics(session['q | PQ110 |




From sample-callflow.csv

| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 55 | IVR300 | MENU | eval:util.getDynamicPrompt("tts","DM100") | eval:util.getDynamicMenu("DM100") |

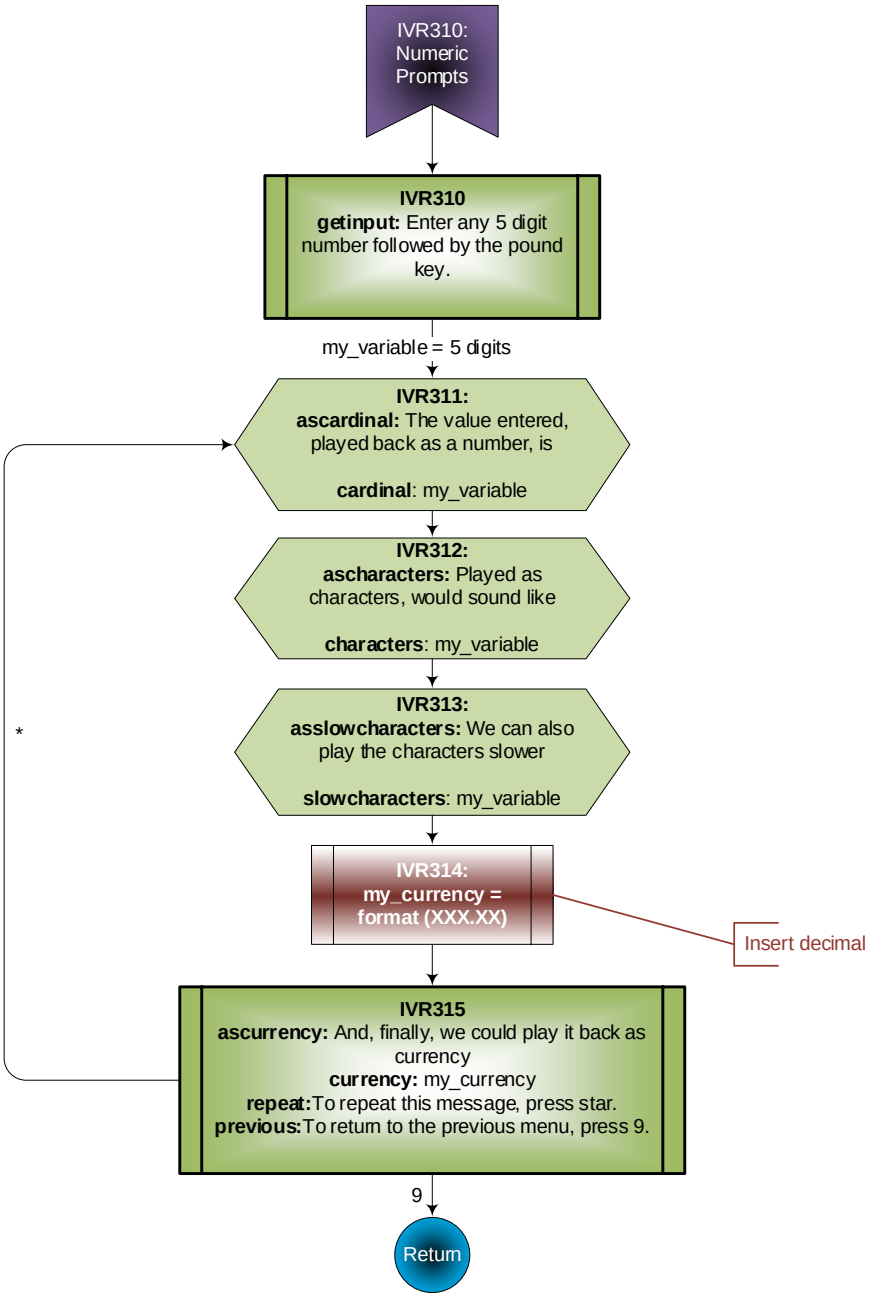
sample-sysadmin.csv – all options enabled

| | A | B |
|---|---|---|
| 1 | KEY | VALUE |
| 7 | DM100 | 1:IVR310\|2:IVR320\|3:IVR330\|4:IVR340\|0:PQ100 |

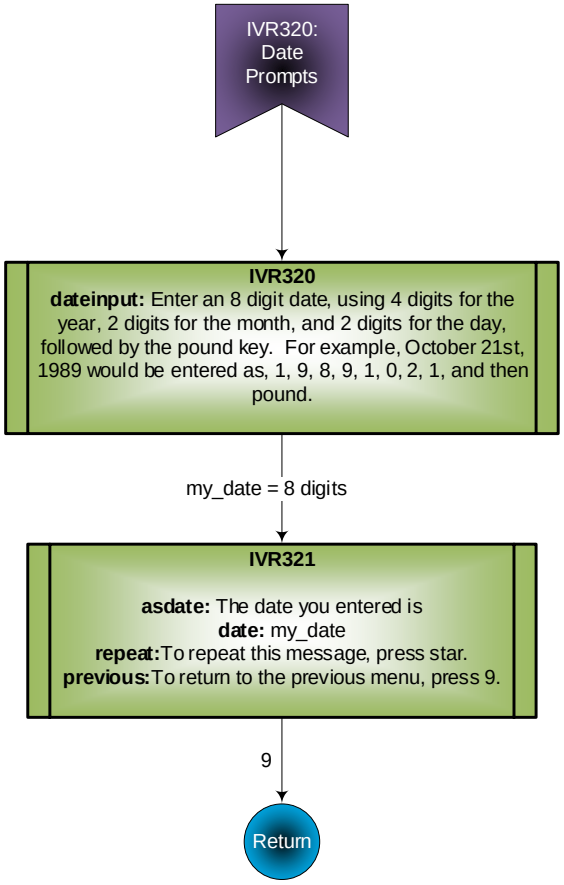
sample-sysadmin.csv – options 3 & 4 disabled

| | A | B |
|---|---|---|
| 1 | KEY | VALUE |
| 7 | DM100 | 1:IVR310\|2:IVR320\|3x:IVR330\|4x:IVR340\|0:PQ100 |

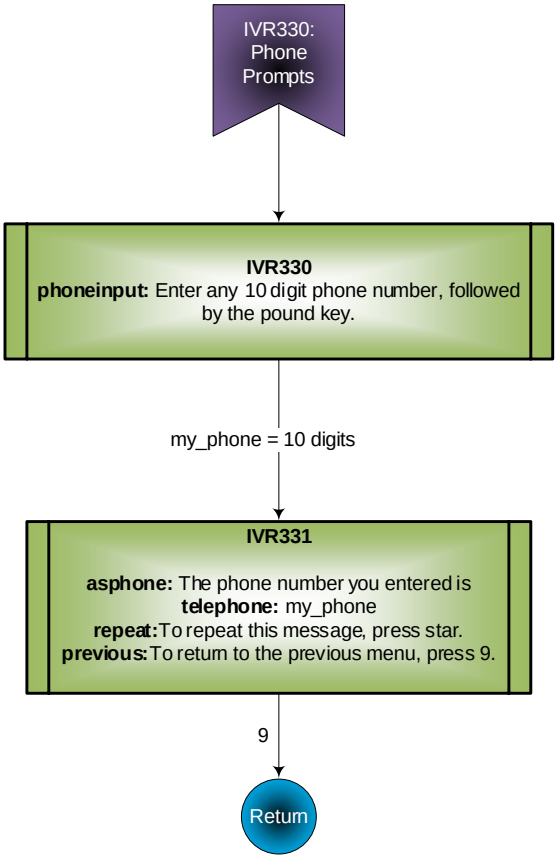
## NUMERICPROMPTS




| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 11 | IVR310 | INPUT | tts:getinput | var:my_variable\|length:5\|action:IVR311\|seconds:3 |
| 12 | IVR311 | PLAY | tts:ascardinal\|cardinal:session["my_variable"] | IVR312 |
| 13 | IVR312 | PLAY | tts:ascharacters\|characters:session["my_variable"] | IVR313 |
| 14 | IVR313 | PLAY | tts:asslowcharacters\|slowcharacters:session["my_variable"] | IVR314 |
| 15 | IVR314 | EXEC | session['my_currency'] = session['my_variable'][:3] + '.' + session['my_variable'][3:] | IVR315 |
| 16 | IVR315 | MENU | tts:ascurrency\|currency:session["my_currency"]\|tts:repeat\|tts:previous | *:IVR311 |

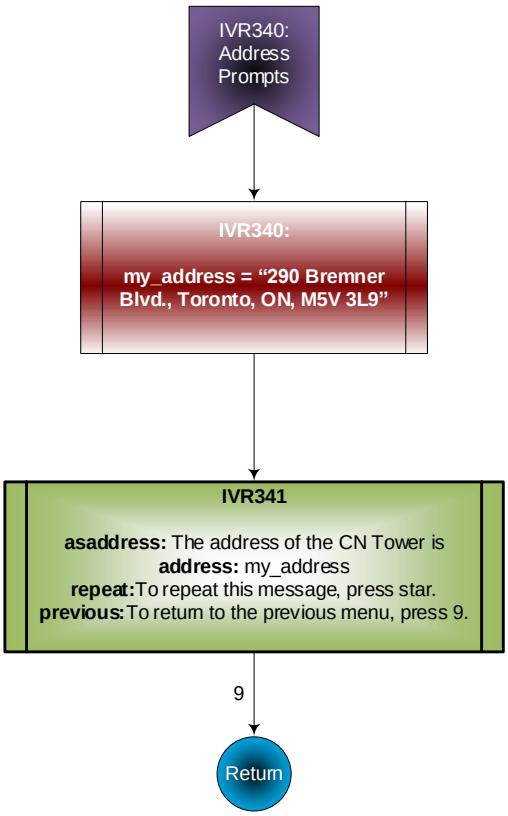



| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 17 | IVR320 | INPUT | tts:dateinput | var:my_date\|length:8\|action:IVR321\|seconds:3 |
| 18 | IVR321 | MENU | tts:asdate\|date:session["my_date"]\|tts:repeat\|tts:previous | |

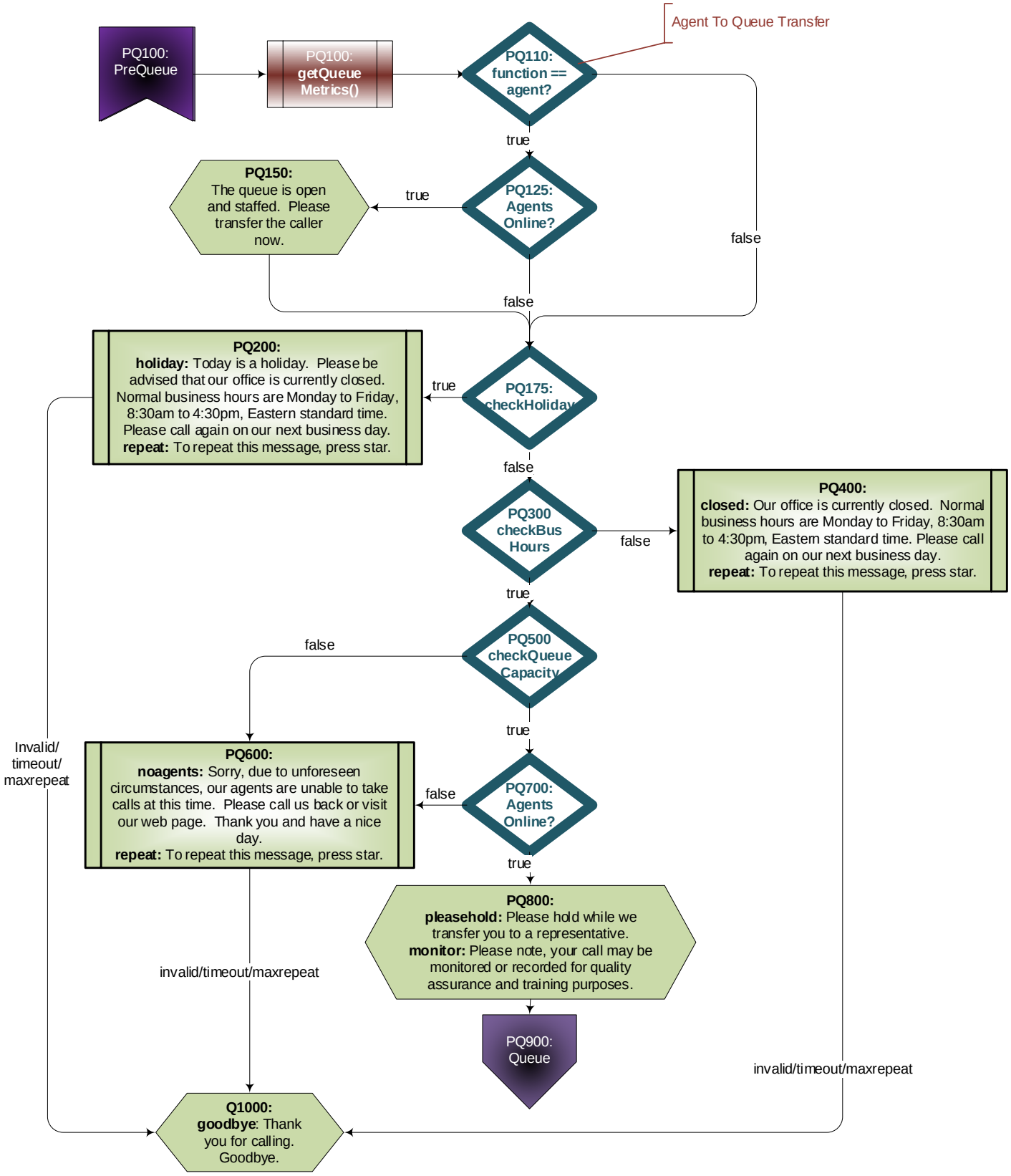


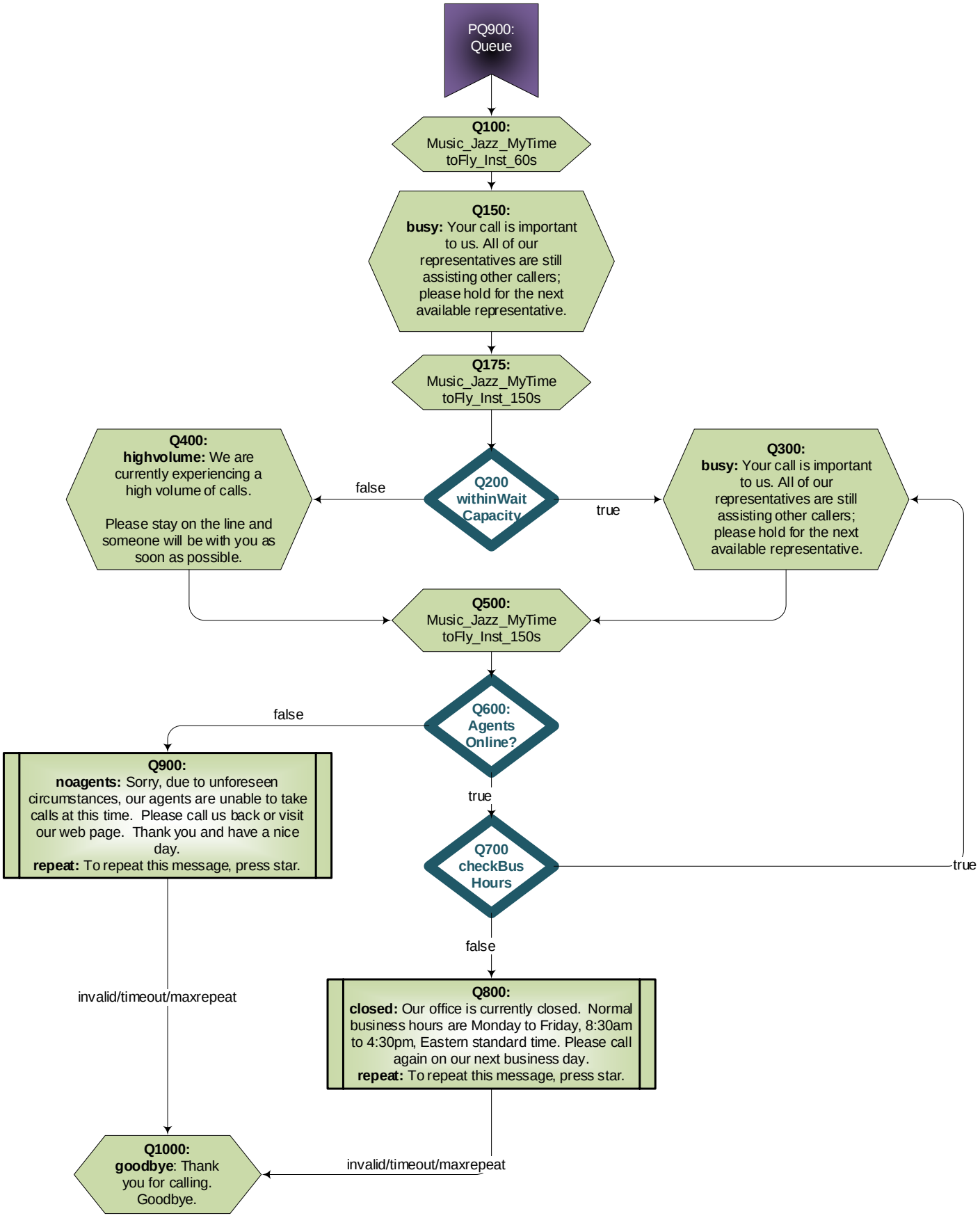

| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 19 | IVR330 | INPUT | tts:phoneinput | var:my_phone\|length:10\|action:IVR331\|seconds:3 |
| 20 | IVR331 | MENU | tts:asphone\|telephone:session["my_phone"]\|tts:repeat\|tts:previous | |

## ADDRESSPROMPTS




| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 21 | IVR340 | EXEC | session["my_address"] = "290 Bremner Blvd, Toronto, ON M5V 3L9" | IVR341 |
| 22 | IVR341 | MENU | tts:asaddress\|address:session["my_address"]\|tts:repeat\|tts:previous | |


PQ100: PreQueue
PQ100: getQueue Metrics()
PQ110: function == agent?
Agent To Queue Transfer
true
false
PQ125: Agents Online?
true
PQ150: The queue is open and staffed. Please transfer the caller now.
false
PQ175: checkHoliday
true
PQ200: holiday: Today is a holiday. Please be advised that our office is currently closed. Normal business hours are Monday to Friday, 8:30am to 4:30pm, Eastern standard time. Please call again on our next business day. repeat: To repeat this message, press star.
false
PQ300 checkBus Hours
false
PQ400: closed: Our office is currently closed. Normal business hours are Monday to Friday, 8:30am to 4:30pm, Eastern standard time. Please call again on our next business day. repeat: To repeat this message, press star.
true
PQ500 checkQueue Capacity
false
true
PQ700: Agents Online?
false
PQ600: noagents: Sorry, due to unforeseen circumstances, our agents are unable to take calls at this time. Please call us back or visit our web page. Thank you and have a nice day. repeat: To repeat this message, press star.
true
PQ800: pleasehold: Please hold while we transfer you to a representative. monitor: Please note, your call may be monitored or recorded for quality assurance and training purposes.
PQ900: Queue
Invalid/ timeout/ maxrepeat
invalid/timeout/maxrepeat
invalid/timeout/maxrepeat
Q1000: goodbye: Thank you for calling. Goodbye.


PQ900:
Queue
Q100:
Music_Jazz_MyTime
toFly_Inst_60s
Q150:
busy: Your call is important to us. All of our representatives are still assisting other callers; please hold for the next available representative.
Q175:
Music_Jazz_MyTime
toFly_Inst_150s
Q400:
highvolume: We are currently experiencing a high volume of calls.
Please stay on the line and someone will be with you as soon as possible.
false
Q200
withinWait
Capacity
true
Q300:
busy: Your call is important to us. All of our representatives are still assisting other callers; please hold for the next available representative.
Q500:
Music_Jazz_MyTime
toFly_Inst_150s
Q600:
Agents
Online?
false
true
Q900:
noagents: Sorry, due to unforeseen circumstances, our agents are unable to take calls at this time. Please call us back or visit our web page. Thank you and have a nice day.
repeat: To repeat this message, press star.
Q700
checkBus
Hours
true
false
Q800:
closed: Our office is currently closed. Normal business hours are Monday to Friday, 8:30am to 4:30pm, Eastern standard time. Please call again on our next business day.
repeat: To repeat this message, press star.
invalid/timeout/maxrepeat
invalid/timeout/maxrepeat
Q1000:
goodbye: Thank you for calling. Goodbye.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | BLOCK | TYPE | PARAM | ACTION |
| 25 | PQ100 | EXEC | session['queuemetrics'] = queues.getQueueMetrics(session['q | PQ110 |
| 26 | PQ110 | EVAL | session['function']=='agent' | true:PQ125\|false:PQ175 |
| 27 | PQ125 | EVAL | session['queuemetrics']['agentsOnline'] > 0 | true:PQ150\|false:PQ175 |
| 28 | PQ150 | PLAY | text:The queue is open and staffed. Please transfer the caller r | PQ175 |
| 29 | PQ175 | EVAL | queues.checkHoliday() | true:PQ200\|false:PQ300 |
| 30 | PQ200 | MENU | tts:holiday\|tts:repeat | timeout:Q1000\|invalid:Q1000\|maxaction:Q1000 |
| 31 | PQ300 | EVAL | queues.checkBusinessHours() | true:PQ500\|false:PQ400 |
| 32 | PQ400 | MENU | tts:closed\|tts:repeat | timeout:Q1000\|invalid:Q1000\|maxaction:Q1000 |
| 33 | PQ500 | EVAL | session['queuemetrics']['contactsInQueue'] < 30 | true:PQ700\|false:PQ600 |
| 34 | PQ600 | MENU | tts:noagents\|tts:repeat | timeout:Q1000\|invalid:Q1000\|maxaction:Q1000 |
| 35 | PQ700 | EVAL | session['queuemetrics']['agentsOnline'] > 0 | true:PQ800\|false:PQ600 |
| 36 | PQ800 | PLAY | tts:pleasehold\|tts:monitor | PQ900 |
| 37 | PQ900 | QUEUE | | Q100 |
| 38 | Q100 | PLAY | wav:Music_Jazz_MyTimetoFly_Inst_60s | Q150 |
| 39 | Q150 | PLAY | tts:busy | Q175 |
| 40 | Q175 | PLAY | wav:Music_Jazz_MyTimetoFly_Inst_150s | Q200 |
| 41 | Q200 | EVAL | queues.withinWaitCapacity(300) | true:Q300\|false:Q400 |
| 42 | Q300 | PLAY | tts:busy | Q500 |
| 43 | Q400 | PLAY | tts:highvolume | Q500 |
| 44 | Q500 | PLAY | wav:Music_Jazz_MyTimetoFly_Inst_150s | Q600 |
| 45 | Q600 | EVAL | queues.agentsOnline() | true:Q700\|false:Q900 |
| 46 | Q700 | EVAL | queues.checkBusinessHours() | true:Q300\|false:Q800 |
| 47 | Q800 | MENU | tts:closed\|tts:repeat | timeout:Q1000\|invalid:Q1000\|maxaction:Q1000 |
| 48 | Q900 | MENU | tts:noagents\|tts:repeat | timeout:Q1000\|invalid:Q1000\|maxaction:Q1000 |
| 49 | Q1000 | PLAY | tts:goodbye | DISCONNECT |
| 50 | | | | |

# Chapter 10: Support

For Additional Support, contact CompuCom at:

**Julius Malixi**
Lead IVR Consultant
CompuCom Canada
(416) 848-0094
julius.malixi@compucom.com

**Mitesh Bhavsar**
Information Technology Manager
CompuCom Canada
(647) 290-6975
mitesh.bhavsar@compucom.com