Contact Center IVR Development Guide

February 2024

Revision History

Date	Description
01.10.2014	First version of the document
22.12.2020	Rebranded for Sinch
22.09.2021	Checked and updated content for validity and marked parts that only concern on-premise installations
08.02.2024	Reformulated note on using Python customizers in Sinch Contact Pro cloud.

Table of Contents

About this Document

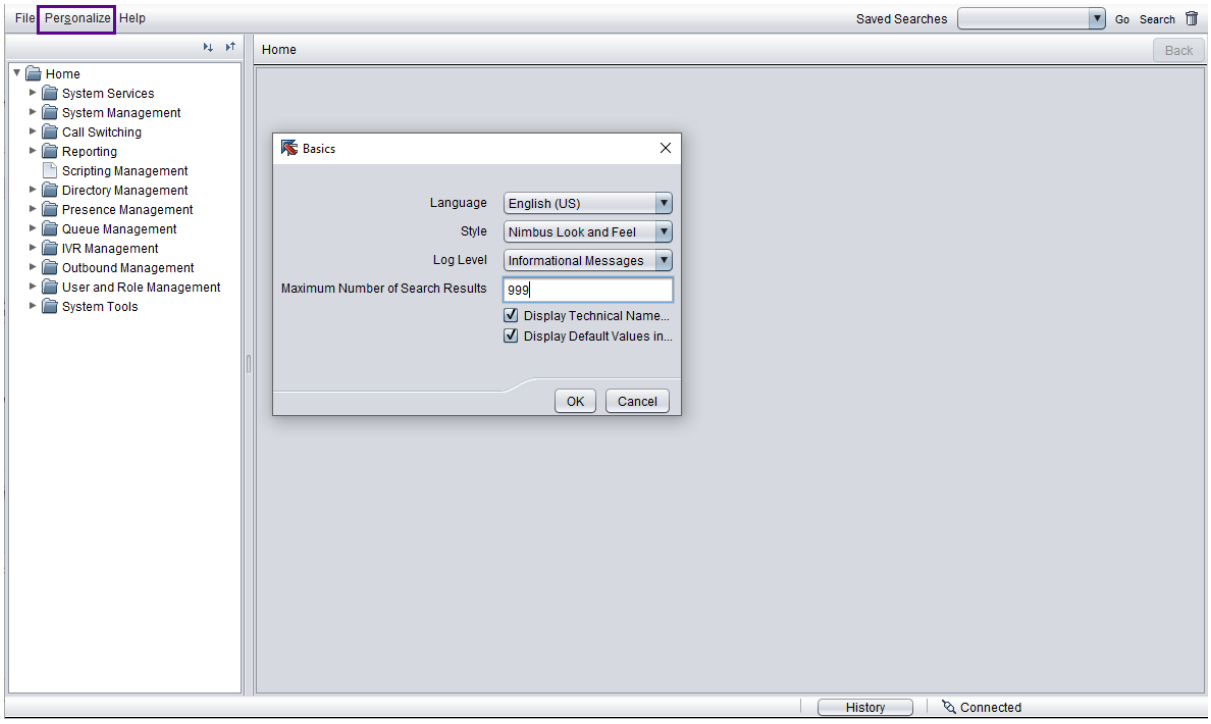
This document is an extension to the System Configurator guide and is aimed at administrators who create IVR applications in System Configurator (SC).

NOTE Your SC application may appear different from the screenshot examples because of the used version and language.

1 Getting Started

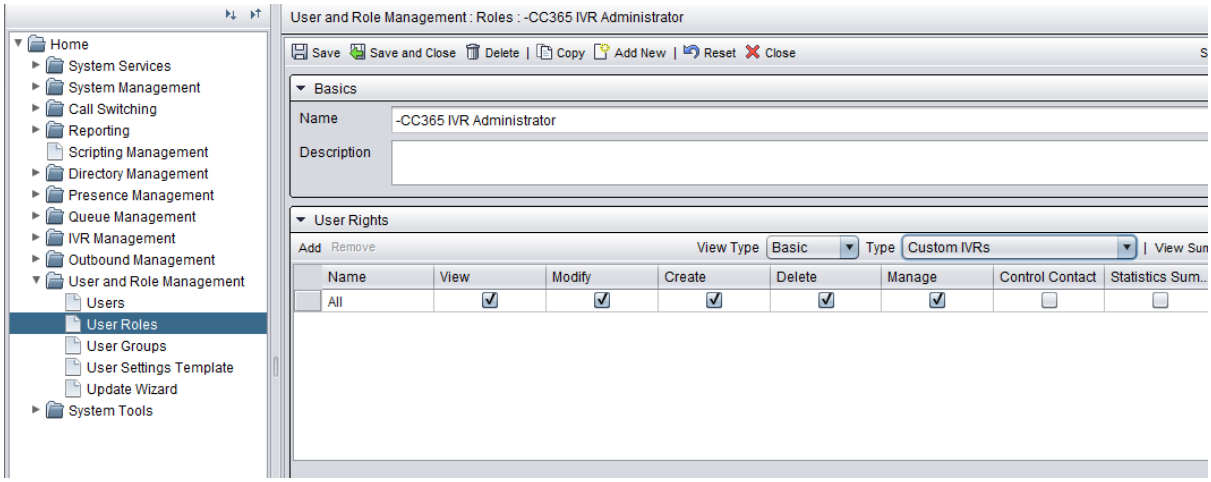
1.1 Personalizing System Configurator

In larger installations, you may want to change the maximum number of search results returned, especially when you do not enter a search term.



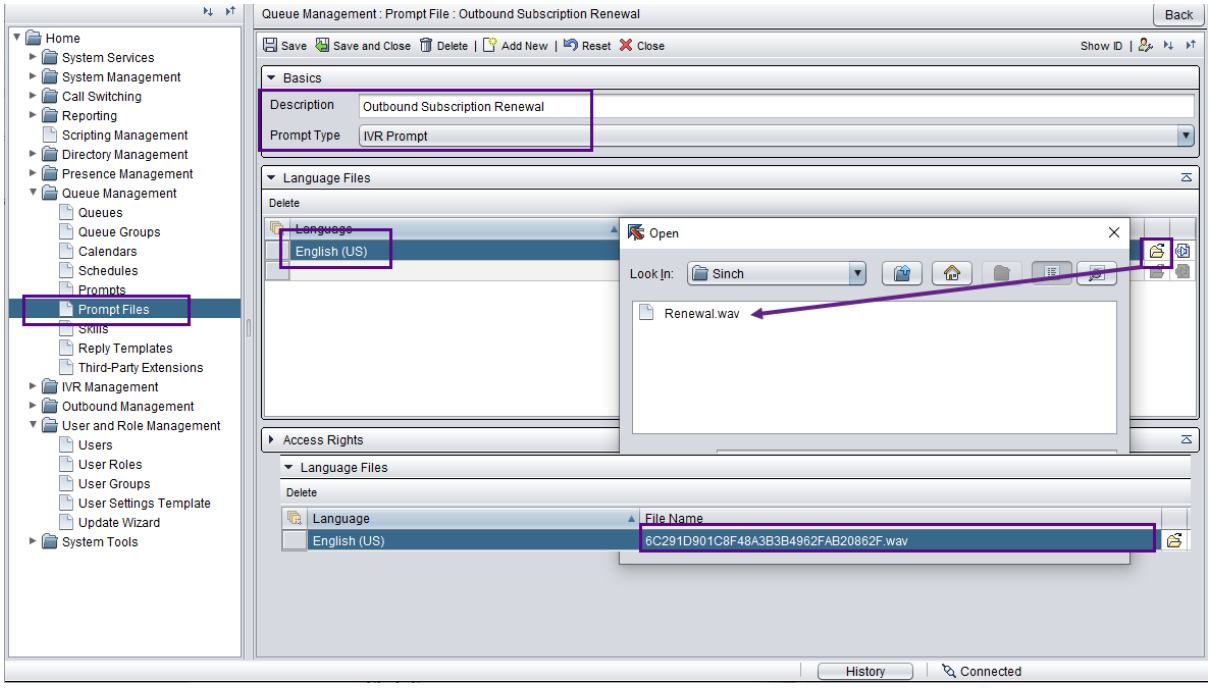
1.2 User Rights

You must have rights to create, view and modify custom IVRs.



1.3 Configuring Customized IVR Prompts

1. In on-premise environments, place your customized prompt file in any folder in the Sinch Contact Center server. For example: C:\Sinch\ContactCenter\. In cloud environments, start from step 2.
2. Using System Configurator, open *Prompt Files* and add a prompt file to the Sinch Contact Center system:
    1. Select prompt type IVR Prompt.
    2. Select language English (US) because it is the default language. Wav files for other languages can also be added.
    3. Browse to the correct file in the folder:



4. Click **Save**. The file is saved with a GUID name. The system asks if you want to create a prompt connected to the new prompt file.

3. Answer **Yes and Open**.
   The system uses default options to create a prompt from the new prompt file.
4. Accept the new IVR prompt by clicking **Save** (or **Save and Close**).
   After creating the prompt click **Refresh** to see the new prompt in the **Queue Management > Prompts** list.

Note: In the way instructed above, IVR prompt files will be created as GUID into the system. Therefore, prompts cannot be transferred to another Sinch Contact Center system as such, but the above steps must be repeated in the target Sinch Contact Center system.

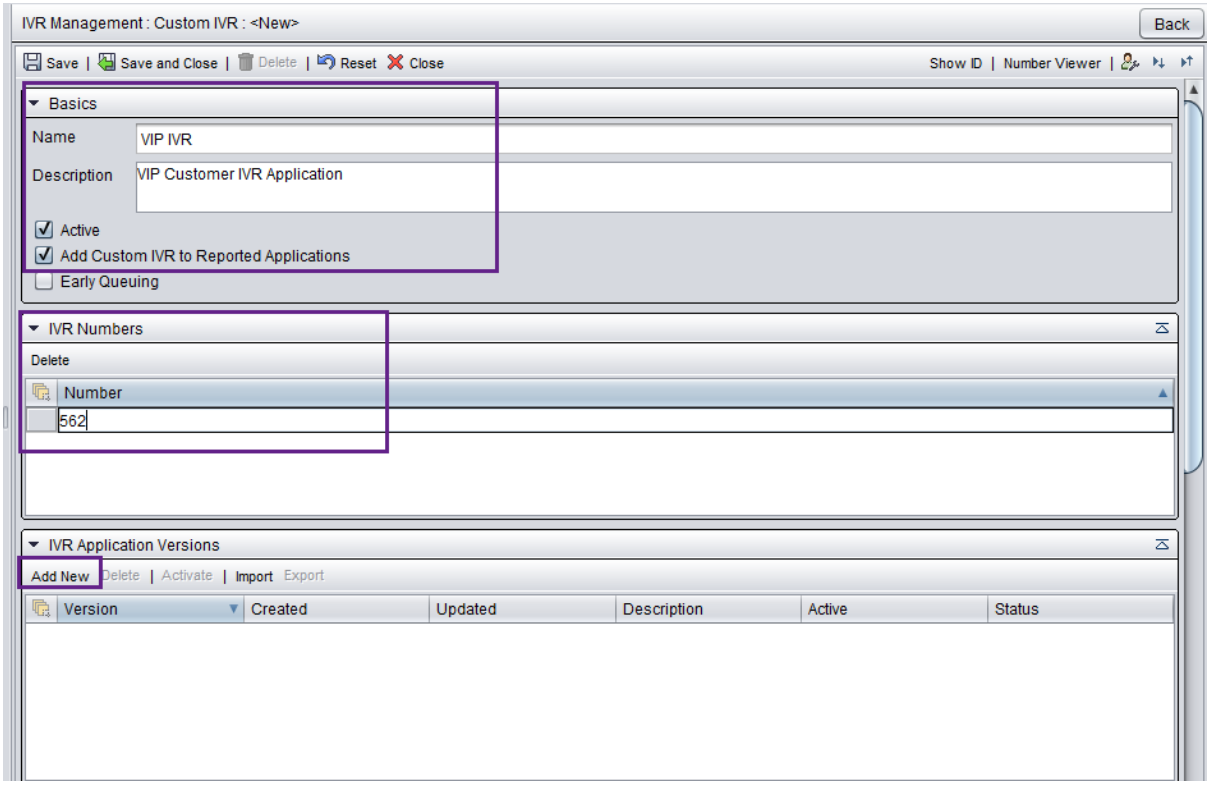
In on-premise environments, the alternative way is to use a folder for storing prompt files:

- The prompt path can be a local path or a shared folder on a file server. For example: \\SERVERNAME\prompts.
- The advantage of a shared folder is that you do not need to copy the prompts to each server when a failover setup with multiple servers is used.
- You can create your own folders using Windows Explorer under this folder (for example, ME_IVRPrompts) and then refer to this folder in your IVRs.

2 Creating an IVR Application

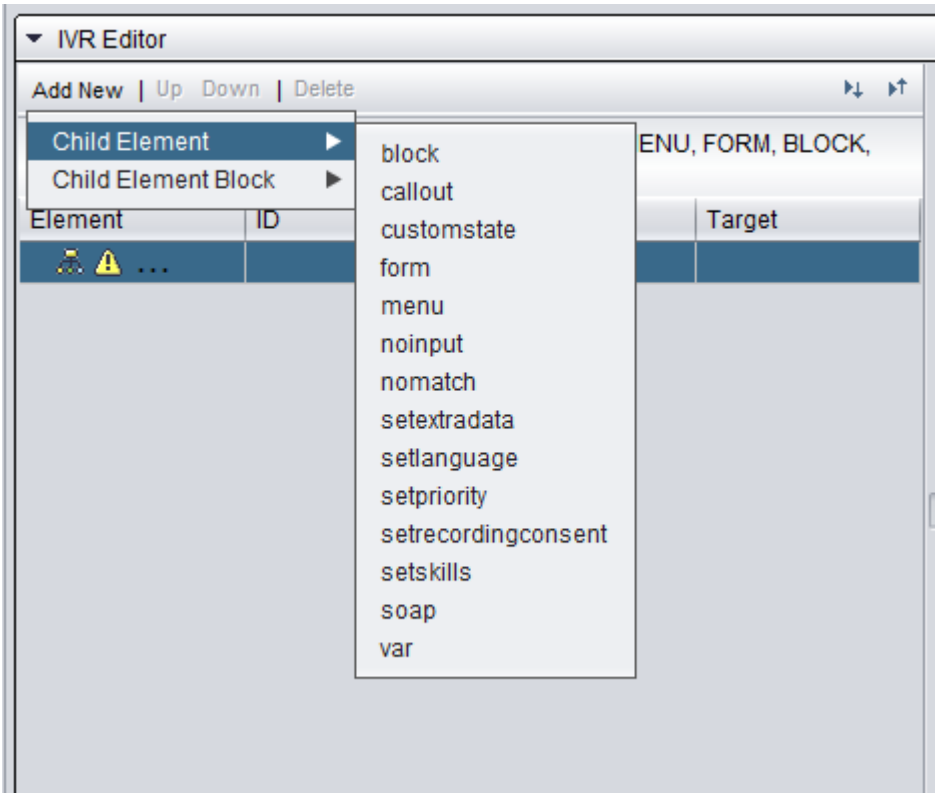
2.1 Add New Application

1. Enter a name and description for the application.
2. Select **Active** and **Add Custom IVR to Reported Applications**, and optionally **Early Queuing**. To learn more about early queuing, see System Configurator document (Call Switching > Global Switching Settings > Managing Signaling > Early Queuing and Toll-Free Queuing).
3. Add IVR number(s).
4. To start creating your new IVR, click *Add New*.



2.2 IVR Editor

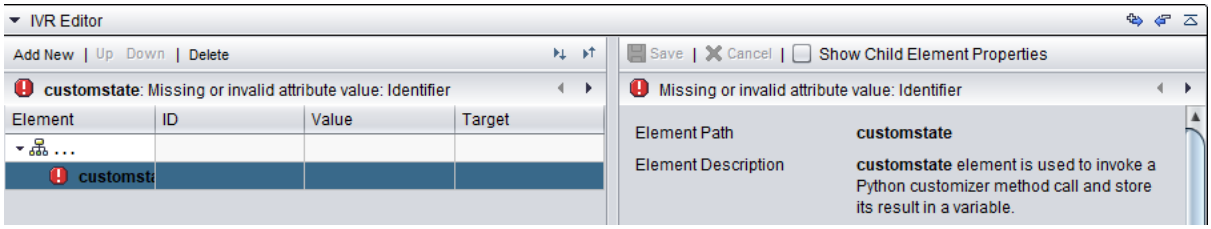
An IVR is made up of elements. Major elements (for example, *block*, *callout*, *customstate*, and *form*) must be given a unique name. To add an element, either click the *Add New* button or right-click on an existing element.



Sinch Contact Center IVR file can be stored, imported, and exported in VoiceXML format. Internally Sinch Contact Center compiles VoiceXML into the Python programming language and uses Python when the IVR application is running. Therefore, values and variables could be written as Python statements.

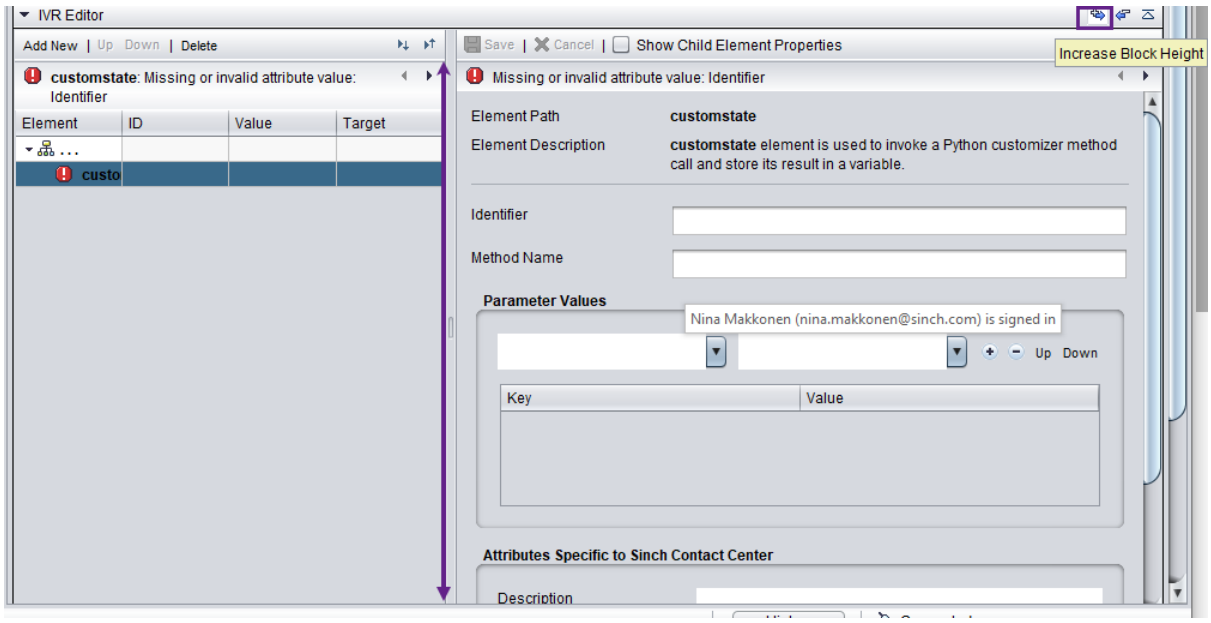
2.3 Adding Information to Elements

Elements are shown in the left frame and their parameters on the right. At least some of the fields must have a value. If a value is missing, a sign is displayed. A message shows what is missing. The sign disappears when all the needed parameters have been entered.



2.4 Maximizing the Work Area

To widen the work area of the IVR Editor, move the divider between the SC menu and the working area as far to the left as possible. To increase the height of the working area, click on the *Increase Block Height* icon.



To expand the elements of the IVR Editor, click on the  icon.

3 VoiceXML Features

3.1 Supported Elements

For a list and detailed information of supported VoiceXML elements and extensions to the VoiceXML standard, see the *System Configurator* guide: *IVR Management > Using IVR Editor > Supported VoiceXML Elements, Element Attributes, Data Items and Queries, and Element Blocks*.

3.2 Commonly Used VoiceXML Elements

For more details on the elements presented here, see the *System Configurator* guide (*IVR Management > Using IVR Editor > Supported VoiceXML Elements*).

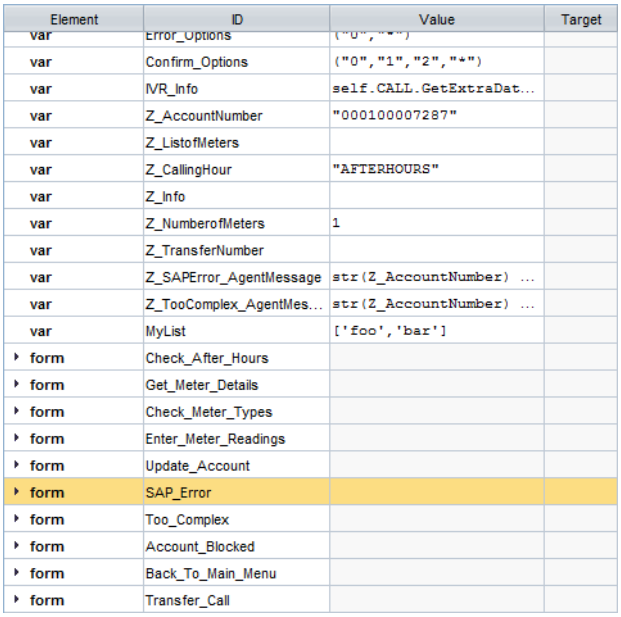
3.2.1 var

- The `var` element is used to store values during IVR execution.
- You should declare all your variables at the start of the vxml document. Variables declared here can be accessed from all subsequent elements.
- Using this element makes the IVR vxml document easier to read.
- To declare variables for a specific element only, declare it in that element, such as `form`.
- You can assign initial values to your variables or use the `assign` element later on.

Element	ID	Value	Target
...			
var	SAPGet_Meter_Details_wsdl	"http://dc2-sapr3-d01...	
var	SAPValidate_Meter_Reads...	"http://dc2-sapr3-d01...	
var	SAPUpdate_Meter_Reads_...	"http://dc2-sapr3-d01...	
var	Blocked_Account_Prompt		
var	Request_Not_Processed_...		
var	Return_Main_Menu_Prompt		
var	Speak_To_Customer_Rep_...		
var	Multiple_Meters_Prompt		
var	Z_IVRAfterHoursResidential	"9001"	
var	Z_IVRInHoursResidential	"9000"	
var	Z_Meter_Counter		
var	Z_Meter_Reading		
var	Z_Meter_Readings	[]	
var	Z_Get_Meter_Details		
var	Z_Meter_Types		
var	Z_QueueResidential		

- Values must be valid python data types.

3.2.2 form



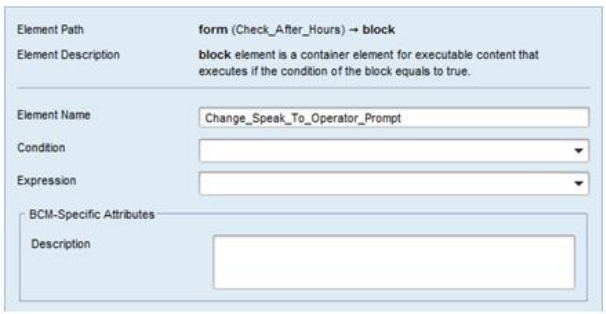
Element	ID	Value	Target
var	Error_Options	("0","*")	
var	Confirm_Options	("0","1","2","*")	
var	IVR_Info	self.CALL.GetExtraDat...	
var	Z_AccountNumber	"000100007287"	
var	Z_ListofMeters		
var	Z_CallingHour	"AFTERHOURS"	
var	Z_Info		
var	Z_NumberofMeters	1	
var	Z_TransferNumber		
var	Z_SAPError_AgentMessage	str(Z_AccountNumber) ...	
var	Z_TooComplex_AgentMes...	str(Z_AccountNumber) ...	
var	MyList	['foo','bar']	
form	Check_After_Hours		
form	Get_Meter_Details		
form	Check_Meter_Types		
form	Enter_Meter_Readings		
form	Update_Account		
form	SAP_Error		
form	Too_Complex		
form	Account_Blocked		
form	Back_To_Main_Menu		
form	Transfer_Call		

- The `form` element is used as a container for other elements.
- The form name is used in goto elements.
- Commonly used child elements of a `form` are:
  - `block`
  - `field`
  - `transfer`
  - `soap`
  - `customstate`
- If you define `var` elements, they can be used within the `form` element only.

3.2.3 block

- The `block` element can be used as a container.
- The block name is used in goto elements.
- A `block` element is required for adding:
  - `assign`
  - `audio`
  - `prompt`
  - `if`

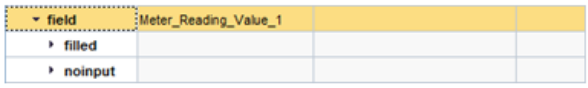
form	Check_After_Hours		
block	Change_Speak_To_Operat...		
if		MyList.count('foo') > 0	
log		DBG> MyList.count('fo...	
assign		""	Speak_T...
assign		("*")	Error_O...

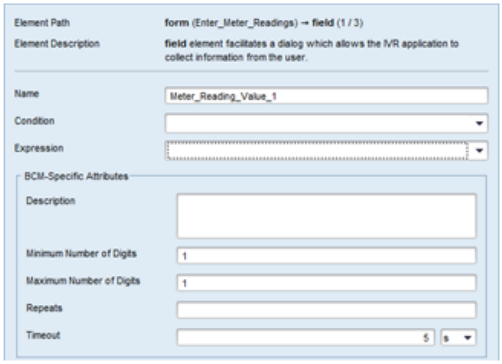


- `log`
- `goto`

- A `block` element can include a condition so that it is only executed if the condition is met (a simple if with no else if or else. In case elseif and else is required Conditional Block will add all elements at once).

3.2.4 field





- The `field` element is used to to collect information from callers.
- It can be used for simple menu types in addition to collecting longer strings of digits.
- `field` elements normally require the following child elements:

  - `audio`

  Announces what is expected from the caller.

  - `filled`

  Executed when the input meets the minimum and maximum

number of digits specified in the field element.

- `noinput`

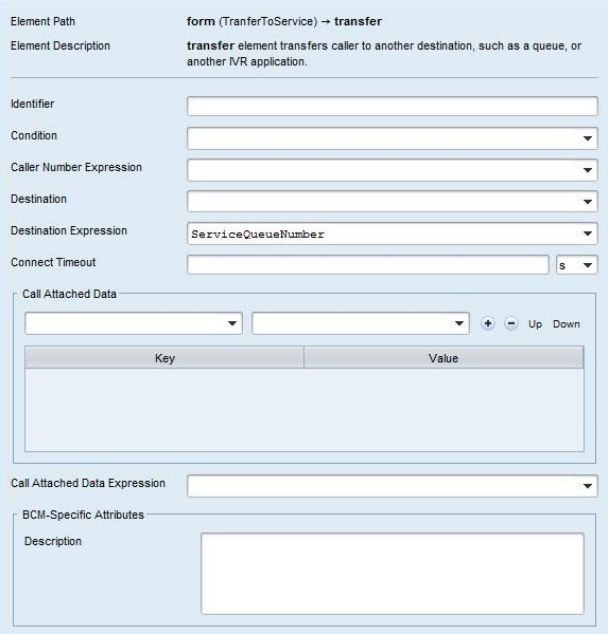
  Executed when the timeout specified in the field element is reached and there has been no input at all.

- `nomatch`

  Executed when the timeout specified in the field element is reached and there has been some input but it does not meet the minimum or maximum number of digits.
  Not required when only 1 digit is expected (as in a simple menu).

3.2.5 transfer

- The `transfer` element is used to transfer the call to another number. This can be:
  - Another IVR
  - A queue
  - A Sinch Contact Center internal extension
  - Any valid external telephone number.

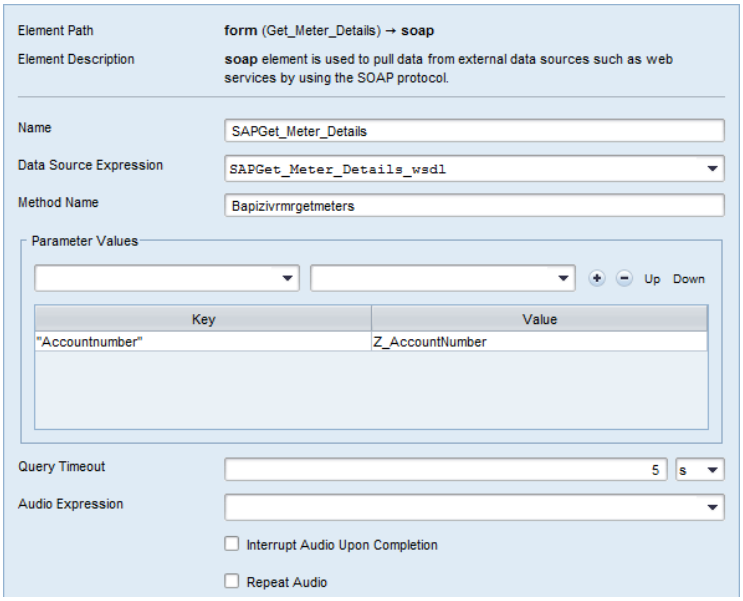


- Destination can be a variable – use Destination Expression.
- Call Attached Data is a string or variable populated in the previous step.
- For complex Call Attached Data strings, it is easier to create this using `customstate`.

3.2.6 soap

Note: In Sinch Contact Pro cloud environments, using SOAP requires making a service request to Sinch support.

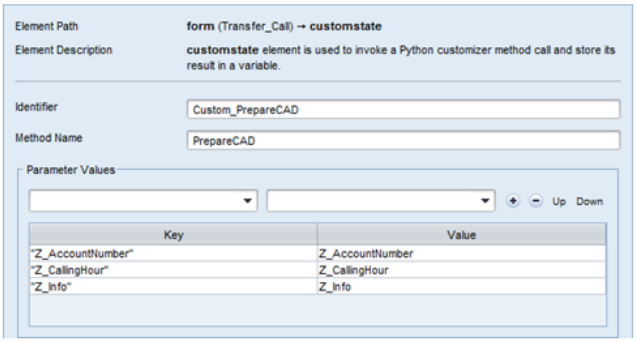
- The `soap` element is used to fetch data from external data sources by using the SOAP protocol.
- It relies on a SOAP or web service defined in a target system such as SAP ERP or SAP CRM.
- In SAP ERP or SAP CRM, these web-services can be a standard ESoA service or a function module exposed as a web-service.
- An end-point needs to be created in SOAPMANAGER for web-services in SAP ERP or SAP CRM.

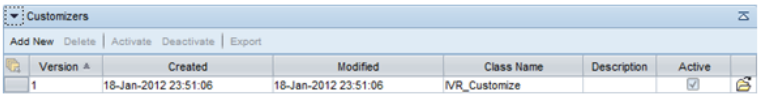


3.2.7 customstate

Note: Using this element is not supported in Sinch Contact Pro cloud environments.

- The `customstate` element is used to execute a python customizer.
- If customized functions are needed to a custom-made IVR, they can be added to the system as Python code. Only one customizer can be active at a time for each custom-made IVR.
- To call the customizer from the IVR application, use the `customstate` element.





3.2.7.1 When to use custom functions

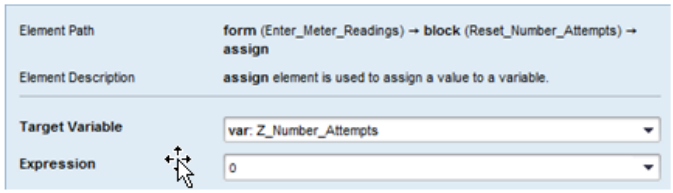
- Custom Functions should only be used if you cannot achieve the same outcome using VXML elements.
- You should be familiar with Python before attempting to create your own custom functions.
- Custom Functions can be used for:
  - OBDC calls to external databases in Sinch Contact Center on-premise environments

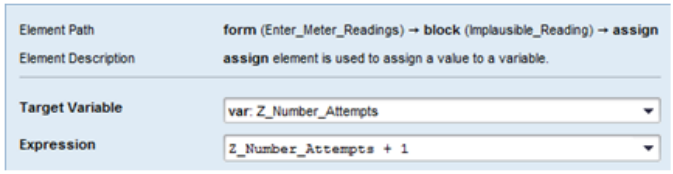
- Doing several looping operations and condition tests to create a python list or dictionary
- SOAP queries if you wish to have the username and password as variables.

- A Python customizer provides a powerful tool to manipulate the Sinch Contact Center system, and therefore you should only use customizers you can trust. Never use customizers from an unknown source, and test all customizers in a test system before use.

3.2.8 assign

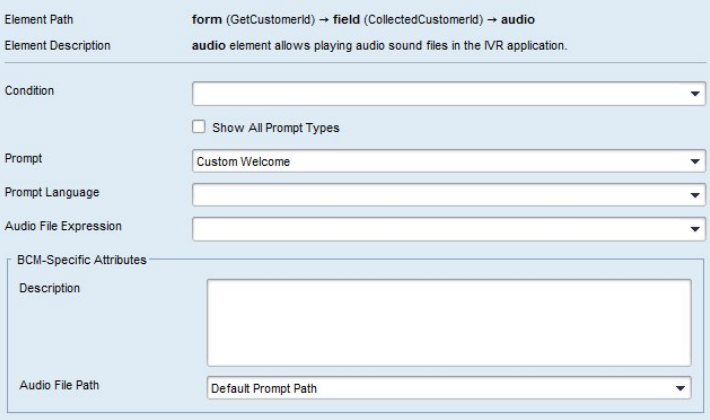
- The `assign` element is used to assign a value to a variable. It is recommended to create all variables first.
- Values must be valid python data types:
  - String
  - Number
  - Dictionary
  - List
- You can assign fixed values.
- You can update existing values:
  - Increase a counter value
  - Add a value to a list.





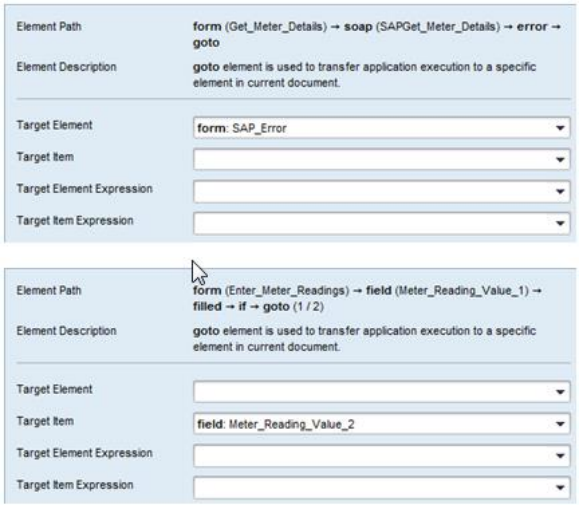
3.2.9 audio

- The `audio` element is used to play audio files (also often referred to as prompts but not to be confused with the prompt element) in the IVR application.
- You can select predefined IVR type Prompt files from a dropdown menu. See how to configure customized IVR prompts in the "Getting Started" section.



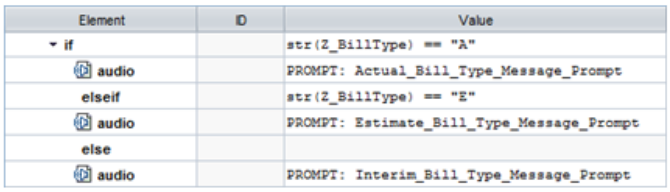
3.2.10 goto

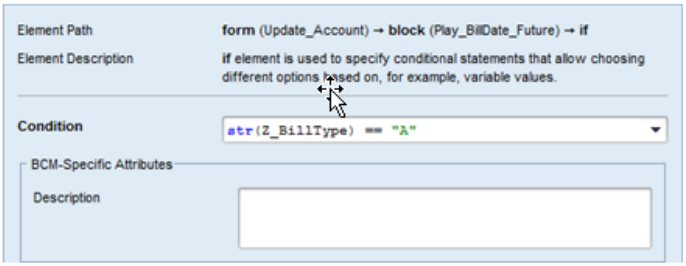
- The `goto` element is used to transfer application execution to a specific element in the current VXML document.
- You transfer to either of the following:
  - an element in the VXML root (normally a `form` element)
  - an item within a root element (normally a `field` or `block` element). Try to keep transfers using this method within the current `form` element so as to make the VXML more readable.



3.2.11 if, elseif & else

- The `if` element is used to specify conditional statements that allow choosing different options based on, for example, variable values.
- You need to specify at least one subsequent element (for example, `assign`, `goto`, `audio`, or `prompt`).
- `elseif` and `else` are regarded as child elements of the `if` element.
- Subsequent elements of `elseif` and `else` do not appear as child elements but act in that way.
- Try to design your IVR so that a condition is always matched (either `if`, `elseif` or `else`).



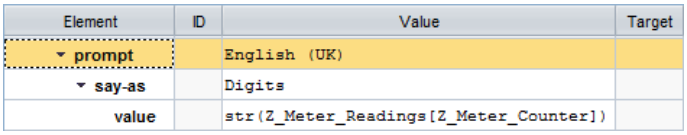


Note: The application evaluates the condition in `if` and `elseif` elements when executing child

elements in the `if` block. If you have multiple child elements in `if` or `elseif` elements, do not use conditions that can change during execution. For example, instead of using queue queries in a condition, use variables where you assign the value before an `if` block.

3.2.12 prompt

- The `prompt` element is used to build audio messages that include numerical information, such as cardinal and ordinal numbers, dates, and times.
- You need to specify the language used.
- You need to add at least one say-as child element.
- In the say-as element, you need to specify:
  - Data Type (Digits, Number, Ordinal, Date, Date and Time or Time)
  - Gender (Feminine or Masculine)
  - Declension (Nominative, Genitive – only required for certain languages e.g. Finnish)
  - A value child element to the say-as element.
- Values must be valid Python data types or variables declared and assigned earlier.



Element	ID	Value	Target
prompt		English (UK)	
say-as		Digits	
value		str(Z_Meter_Readings[Z_Meter_Counter])	

3.3 Call Attached Data

- Call-attached data is nothing more than a Python dictionary (see chapter 6.3).
- You can add any information gathered in the IVR to this dictionary. For example:

```
{"Z_CADAlert": Caller selected Support – Account Inquiries,
"Z_AccountNumber": 123456}
```

- The OII Integrations interface of Sinch Contact Center converts this dictionary into an xml message using the Application ID specified in the OII setup (default is SAP_BCM/OII). For example:

```
<ItemAttachedData><Application
id="SAP_BCM/OII"><FirstBName>FE0CBDF0-FBB6-4FF3-8325-
C77322A15FAD </FirstBName><Z_CADAlert>Caller selected
Support – Account
Inquiries</Z_CADAlert><Z_AccountNumber>123456
</Z_AccountNumber><BNumberName>9216C832-FF83-4DF8-8A1A-
333CD34FAF76</BNumberName>
<FirstANumber>15136023488</FirstANumber><OrigQueue>9216C832-
FF83-4DF8-8A1A-333CD34FAF76
</OrigQueue></Application></ItemAttachedData>
```

- If you integrate Sinch Contact Center with SAP CRM, you may wish to send data to an additional Application ID: CRM_IC/BUPA.
  - Account identification in SAP CRM IC WebClient is done using a phone number or GUID of a Business Partner. This GUID has to be sent to the Application ID: CRM_IC/BUPA.
  - To automatically confirm a Business Partner, an ‘X’ needs to be sent using the key BPCONFIRMED in the same Application ID.
  - All other information sent using this Application ID can be overwritten by SAP CRM and therefore lost when the call is transferred.
- If you integrate Sinch Contact Center with SAP CRM or SAP ERP using SAPphone, you may wish to send data to an additional Application ID:

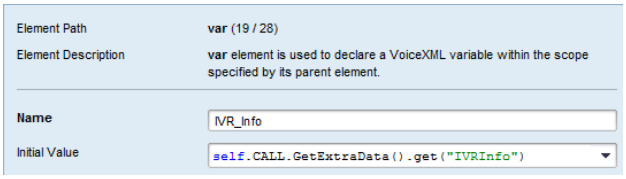
SAPphone.

Sinch Contact Center OII converts the XML data to send to SAPphone

- In order to use different Application IDs, you need to create your own xml string. The customizer highlighted in chapter 3.2.7 does this for you:

```
<Application
id="CRM_IC/BUPA"><CURRENTCUSTOMER></CURRENTCUSTOMER><CURRENT
CONTACT>
</CURRENTCONTACT><BPCONFIRMED></BPCONFIRMED></Application>
<Application id="SAPphone"><Calldata obj="KEYVALUE"
inst="0001" key="Z_AccountNumber">123456</Calldata>
</Application>
```

  - CAD passed to your IVR from another IVR is held in the dictionary "IVRInfo“



    - To retrieve the value of Z_AccountNumber you can use:

- self.CALL.GetExtraData().get(“IVRInfo”,{}).get(“Z_AccountNumber”)

4 Best Practices for IVRs

Declare any information you might want to change or use later as a variable. These include, for example:

- Prompt files and collections of different prompt files
- Soap wsdl urls
- Valid keys to be selected in menus
- Counters
- Information to be sent as CAD.

Map a process into a form. A process can be, for example:

- Asking the caller for input and checking the result
- Calling a Soap query and then checking the result
- Handling errors and asking the caller for the next step
- Preparing the CAD and transferring the call.

Use fields for all caller input, as they:

- Can be contained in a form (a menu cannot)
- Are more flexible with checking for input and determining the next step.

4.1 Basic Functions (ANumber, BNumber, and so on)

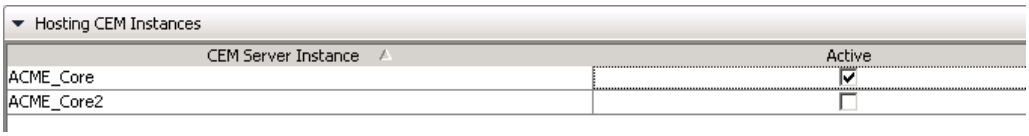
- **DirectoryService-API** (To see how to use this function, see ANumber recognition IVR exercise from the IVR repository).
- **GetNumberInfo** method allows checking Agent status (see Appendix 2).
- **GetCurrentPRSProfile** method is used to check agent profile status.
- **AGENT.FIND** method can be used to search for an agent object based on address info or user GUID.
- **Example of modified** For *Example_IVR_Conditional.xml* and usage of *Queue.Query*, see appendices 3 and 4. For a sample file, contact support.
- **QUEUE.QUERY** API is documented in the System Configurator document: *IVR Management > Using IVR Editor > Element Attributes > Data Items and Queries*.

5 Troubleshooting

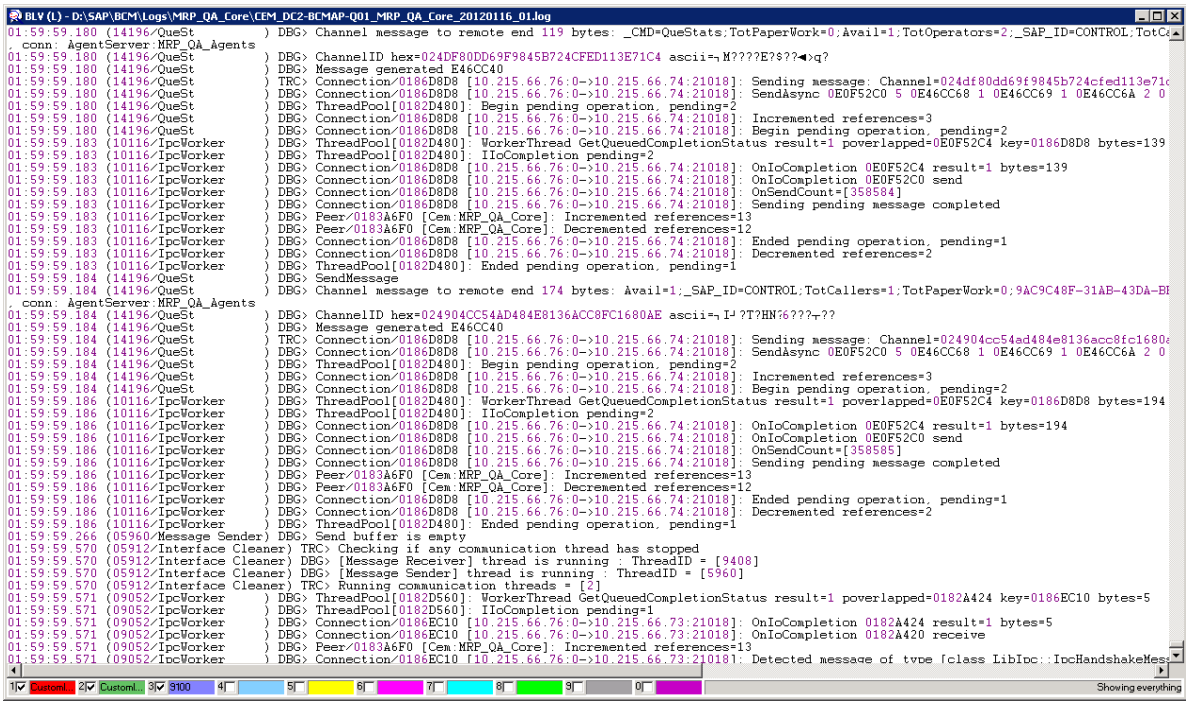
Note: Both the BLV log viewer application and log files are directly only available in Sinch Contact Center on-premise environments. Sinch Contact Pro cloud customers should contact Sinch support with a service request to get access to log files.

5.1 Business Communication Management Log Viewer (BLV)

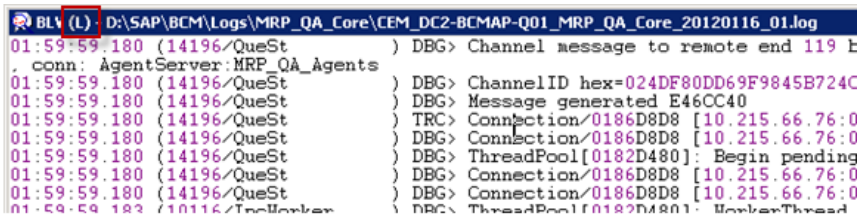
Tip: Use only one Core while debugging an IVR application. This way all events can be collected from a single log:



Sinch Contact Center on-premise software package includes a BLV log viewer application customized for Sinch Contact Center log files:

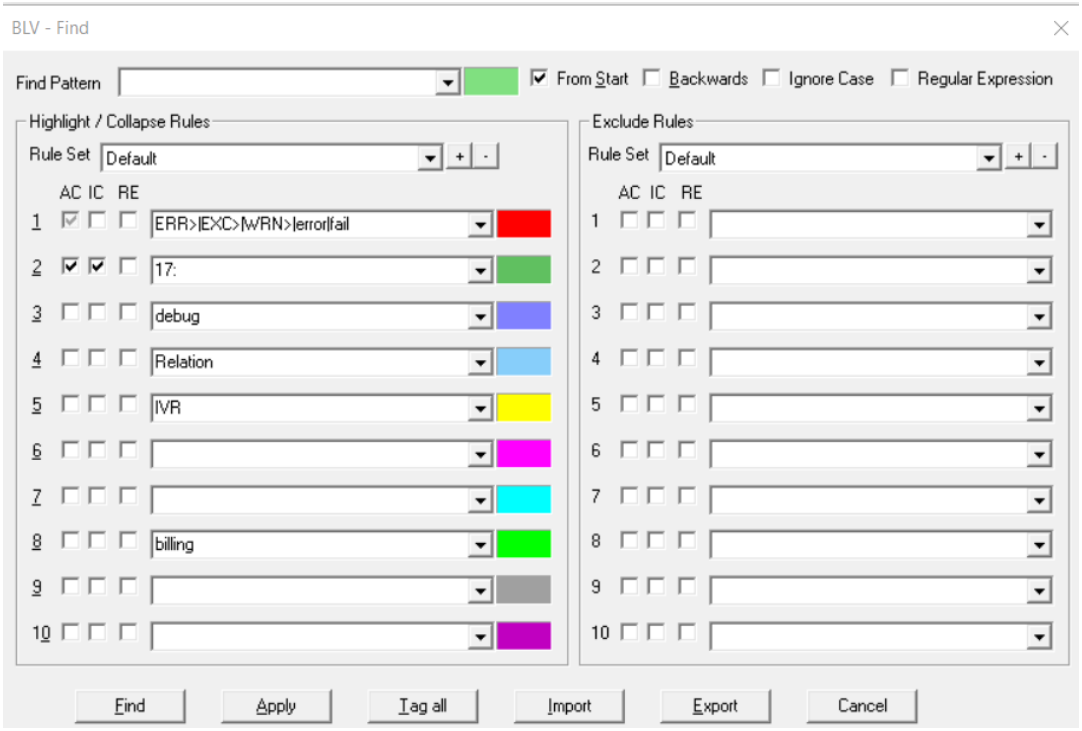


You can view log data updated in real-time in **Live** mode or view the data available at the time you open the log in **Historical** mode. To toggle between the modes, type l (lower case) on your keyboard. Using the up arrow in the file also turns off **Live** mode. Note the (L) appended to BLV in the Windows title bar to indicate **Live** mode:



5.2 BLV Find dialog

Use the **Find** dialog to search the log and highlight matches to your rules in it.
To open the dialog, press Ctrl + F on your keyboard.



You have the following options for finding matches in the log:

- To jump from one match to a pattern to the next, enter the pattern in the **Find pattern** field and click **Find**. This closes the dialog and moves you back to the main screen. You can quickly find the next or previous match with f or F keys.
- To highlight matches or only show lines that include them in the log:

1. Define the necessary rules in the **Highlight / collapse rules** section.

2. Select how you want to be able to see rule matches in the log:

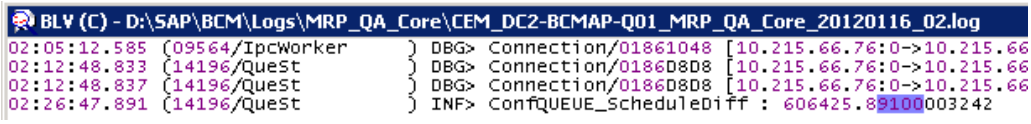
    - To only highlight the matches with a color but show the whole log file, click the **AC** checkbox once on the rule line. This makes the check mark grey.
    - In addition to highlighting, to be able to see only those lines in the log that include these matches, click the **AC** checkbox twice. This makes the check mark black.
    - If you leave the checkbox unselected, the rule will not be used in highlighting.

3. Click **Tag all**. This closes the dialog and tags all lines that match with the currently active Highlight / collapse rules. Tagging is a way of marking lines permanently. In the main screen, you can then jump between them easily (with t and T keys) or filter the view to show only tagged lines (C key).

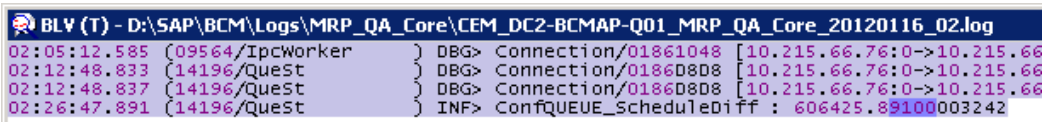
- To filter out noise from a view, enter patterns for this into the **Exclude rules** section. To use the rule in the main screen, press the e key. This hides all lines that contain matches with the currently active Exclude rules.

5.3 Turning Collapsed/Tagged-only mode on and off

Collapsed mode only shows you the lines containing matches to your Highlight / collapse rules that have black check marks (clicked twice). To toggle between **Collapsed** and **Full** modes, press the C key on your keyboard. Note the (C) appended to BLV in the Windows title bar to indicate **Collapsed** mode:

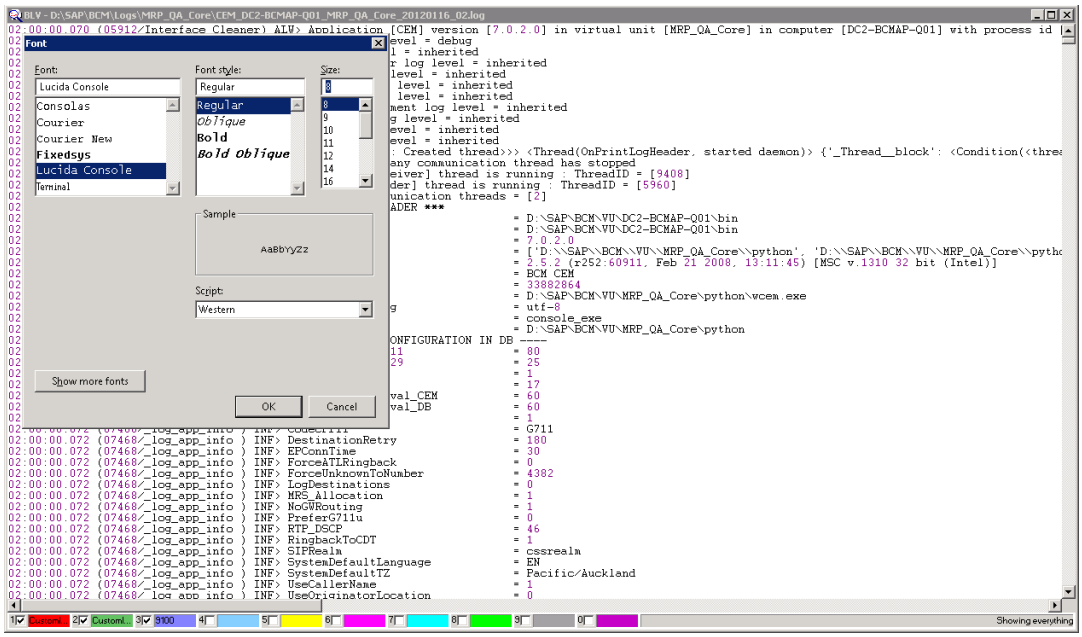


Tagged-only mode doesn't hide any lines. Instead, it tags the lines that include matches to your Highlight / collapse rules and allows you to move from one tagged line to the next. **Tagged-only** mode considers rules with either grey or black check marks (clicked once or twice). To toggle between **Tagged-only** and **Full** modes, type C on your keyboard. Note the (T) appended to BLV in the Windows title bar to indicate **Tagged-only** mode.



Press Ctrl+Shift+T to clear all tags.

5.4 Changing the Font



To open the Font Dialog, type o on your keyboard.

5.5 F1 Help Commands

Alt+T	= Toggle tabs
Ctrl+<Nbr>	+ No selection = Toggle highlight <Nbr>
Ctrl+<Nbr>	+ Selection active = Put selection to highlight <Nbr>
Ctrl+Shift+<Nbr>	+ Selection active = Add selection to highlight <Nbr>
Ctrl+C	= Copy selection to clipboard
Ctrl+F	= Open find dialog or find next occurrence of selected text
Ctrl+N	= Togle number highlighting
Ctrl+S	= Save selection, marked area, matched or tagged lines to file
Ctrl+T	= Tag or untag line
Ctrl+Shift+T	= Clear all tags
Ctrl+WheelUp	= Increase font size
Ctrl+WheelDown	= Decrease font size
c	= Collapse to or uncollapse from matching lines only view
C	= Collapse to or uncollapse from tagged lines only view
e	= Enable or disable excludes
f	= Find next
F	= Find previous
h	= Cycle between normal, hex and hex-with-line-breaks views
l	= Turn live viewing on or off
t	= Goto next tagged line
T	= Goto previous tagged line
m	= Start or cancel area marking
M	= Tag all lines within marked area
o	= Open font selection dialog
s	= Show or hide status line

Down	= Scroll line down
Up	= Scroll line up
Home	= Show start of line
Ctrl+Home	= Goto start of file
End	= Show end of line
Ctrl+End	= Goto end of file
PageUp	= Scroll page up
Ctrl+PageUp	= Change to previous file
PageDown	= Scroll page down
Ctrl+PageDown	= Change to next file
Right	= Scroll column right
Ctrl+Right	= Scroll page right
Left	= Scroll column left
Ctrl+Left	= Scroll page left
Esc	=Cancel selection and instant highlight
Ctrl+Del	= Clear file (close, truncate and reopen)
F1	= Show this help
F3	= Open find dialog
F12	=Toggle allways-on-top mode
Shift+F12	= Cycle through different window transparency levels
Left click	= Start stream selection
Ctrl + Left click	= Start box selection
Left Doubleclick	= Select and instantly highlight word or GUID or text in parens: (...) [...] {...} <...> word = digits letters underscore GUID = 32 hex digits maybe separated with dashes
Right Doubleclick	= Tag or untag line

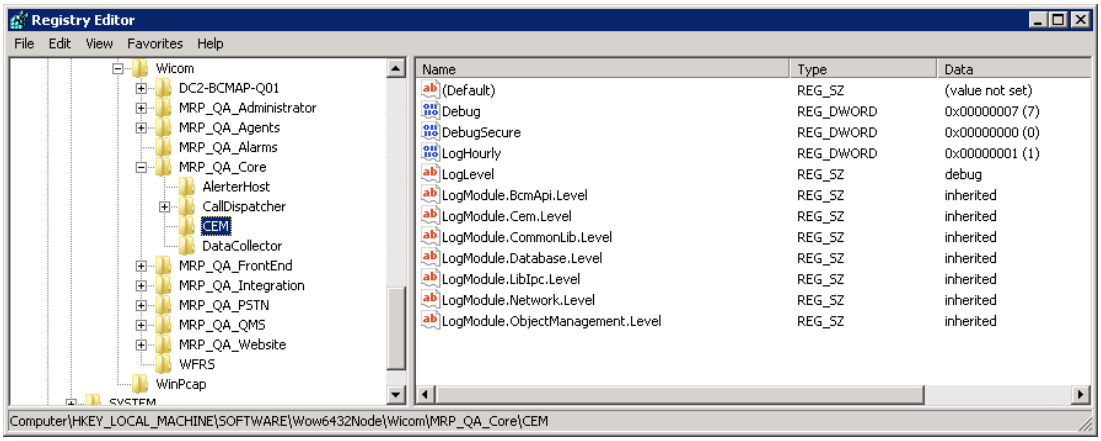
5.6 Debugging your IVR

5.6.1 Setting the CEM log levels to debug

Make sure the CEM log level is high enough to show the IVR detail. In a production environment, do not leave the log levels as Debug after debugging, as the log file size will grow exponentially. Other options are: Warning, Info, and Trace. Warning is usually the default value.

Open Registry Editor on the server and check the following values. Create any missing values:

- Debug = *7*
- DebugSecure = *0*
- LogLevel = *debug*
- LogModule.Cem.Level = *inherited*

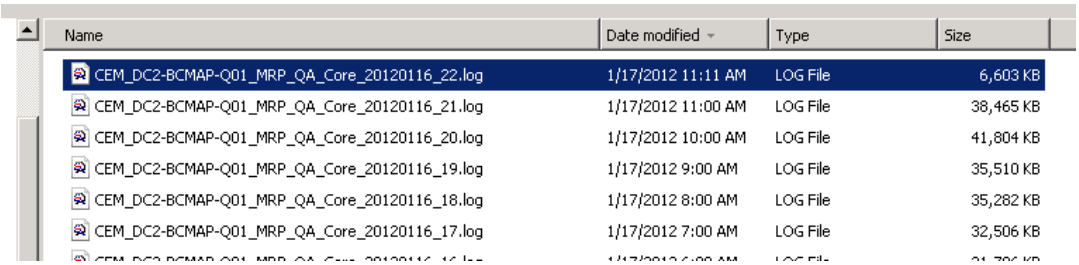


5.6.2 Locating the CEM log

CEM logs are usually located in folder:
drive:\Sinch\ContactCenter\Logs*InstallationName*_Core

Log filename contains the following parts:
CEM_*MachineName*_*InstallationName*_Core_*Date*_*Hour*

Use the latest CEM log for debugging.
Note: Log files use UTC Date and Time regardless of your server time zone settings.
Tip: Sort the files using the Date Modified column in descending order to find the latest file.

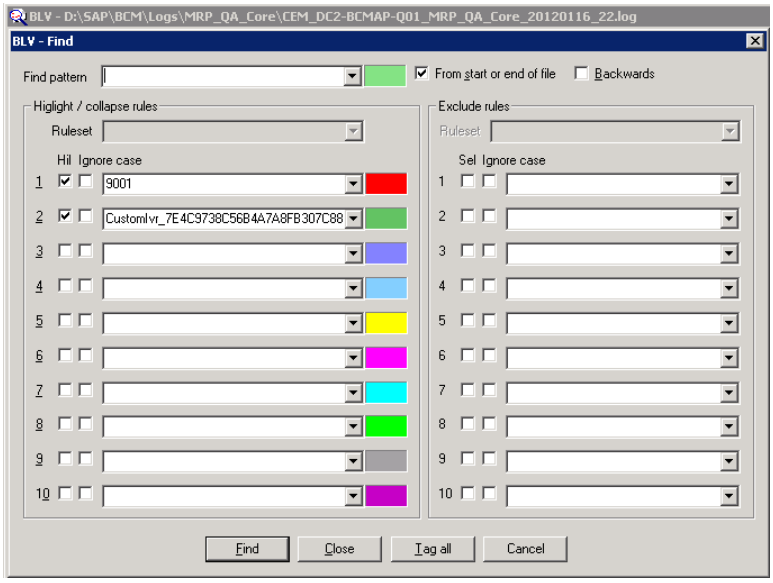


5.6.3 Reading the CEM log

5.6.3.1 Finding log entries related to your IVR

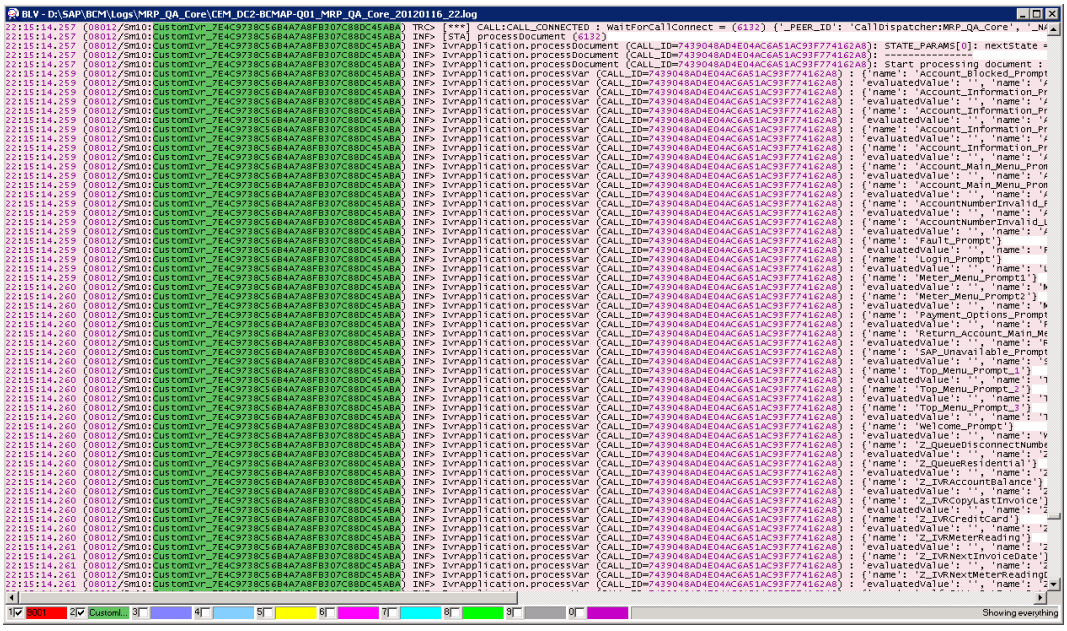
1. Make a call to your IVR and search for the number you called (or called from).
2. Search for the entry ‘CustomIvr_’ after this time.
   Your IVR has a unique GUID (in this case CustomIvr_7E4C9738C56B4A7A8FB307C88DC45ABA).

3. Add this to your search terms in BLV.



4. Click Find.

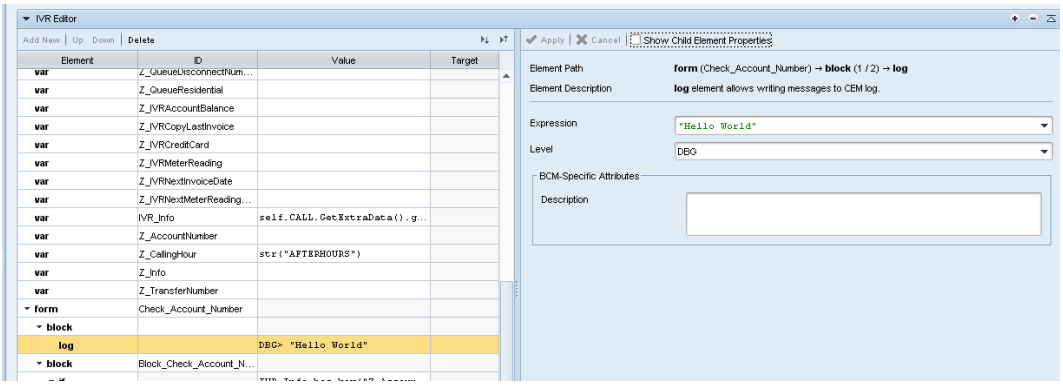
   BLV log viewer shows matching entries:



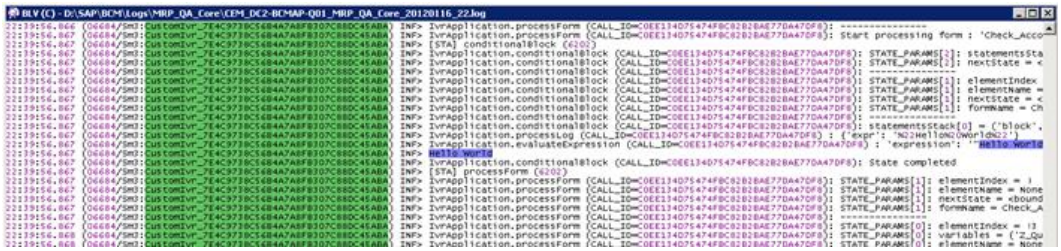
5.6.3.2 Adding log entries into your IVR

There's no breakpoint feature in the IVR. By adding log entries to your IVR, you can find this entry in the CEM log and use it as a reference point. Unique log entries can be used to pin-point each stage. See more about the log element in the *System Configurator* document: *IVR Management > Using IVR Editor > Supported VoiceXML Elements > Log Element*.

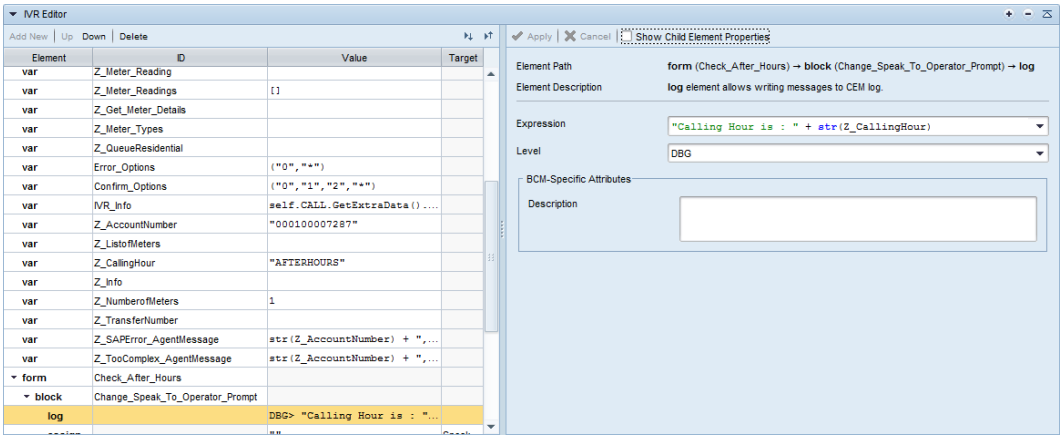
Enter your log entry text to the **Expression** field:



Search results in Core log:



You can also add a log entry to show the current value of a variable, such as *Z_CallingHour*. Enter the name of the variable to the **Expression** field.
To find the variable entry in the log more easily, you can add unique text, such as *Calling hour is*: "Calling Hour is : " + str(Z_CallingHour)



Search results:

6 Python

Note: This chapter is not relevant for Sinch Contact Pro cloud customers.

6.1 Manipulating Strings

Python term	Description	Example	Values in Example	Result
str	String	str(a) + str(b)	a = "Hello ", b = "World"	Hello World
lstrip()	Remove a given character from the left	a.lstrip("0")	a ="000001"	1
split	Split a string at a given character. Produces a List	str(a).split(".")	a = "1.00"	["1", "00"]
replace	Replace a given character in string	a.replace(“o", "")	a = "Hello World"	Hell Wrld
isdigit()	Checks if a string is comprised of just digits	a.isdigit() b.Isdigit()	a = "Hello World" b = “12345"	False True

These functions can be combined: `a.replace("#", "").isdigit()` returns `True` when `a = 12345#`

Python term	Description	Example	Values in Example	Result
len	Length	len(a)	a = "Hello World"	11
	Character at position b	a[b]	a = "Hello World", b = 1	e
	Characters from b to c	a[b:c]	a = "Hello World", b = 1, c=3	el
	Characters up to b	a[:b]	a = "Hello World", b = 3	Hel
	All but characters up to b	a[b:]	a = "Hello World", b = 3	lo World
	Last character	a[-1]	a = "Hello World"	d
	Last but one character	a[-2]	a = "Hello World”	l
	Last b characters	a[-b:]	a = "Hello World", b = 3	rld

	All but the last b characters	a[:-b]	a = "Hello World", b = 3	Hello Wo

6.2 Some other Python expressions used in IVR Editor

Python term	Description	Example	Values in Example	Result
int	Integer	int(a)	a = 1.03	1
datetime.datetime.now()	Returns current date and time in datetime format	datetime.datetime.now()		2012-01-18 10:24:08.259000
datetime.datetime.strptime	Formats a string using a given format in datetime format	datetime.datetime.strptime(a, "%Y-%m-%d")	a = "2012-04-22"	2012-04-22 00:00:00

You can use datetime to check if a date is in the past:

datetime.datetime.strptime(a, "%Y-%m-%d") < datetime.datetime.now()

For more python examples, see:

- http://docs.python.org/tutorial
- Google search: python *term*

6.3 Dictionaries, Lists and Tuples

Dictionaries {}

- A dictionary is an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary).
- Example: {‘First’:1,’Second’:2,’Third’:3,’Fourth’:4}

Lists []

- Lists are a list of values, each one numbered, starting from zero. The first one is numbered 0, the second 1, the third 2, etc. You can remove values from the list and add new values to the end.
- Example: [‘First',‘Second',‘Third',‘Fourth’]

Tuples ()

- Tuples are just like lists, but you can't change their values. Again, each value is numbered starting from zero, for easy reference.
- Example: (’Sunday’,Monday,’,’Tuesday',‘Wednesday',‘Thursday',‘Friday',‘Saturday’)

6.4 Dictionaries, Lists and Tuples – Examples

A simple dictionary:

```
{'Multi': None, 'Meters': None, 'Blocked': 'X'}
```

A dictionary inside a dictionary:

```
{'Return': {'Type': 'E', 'Number': '016', 'Id': 'ZIVRMES'}, 'Multi': None, 'Meters': None, 'Blocked': 'X'}
```

A dictionary inside a tuple:

```
('item', {'Division': '51', 'Installation': '3000194370', 'Register': '001', 'Equipment': '000000000040099113', 'Oldreading': '29165', 'Newreading': None, 'Meternumber': '00000000008000104'})
```

A dictionary inside a tuple, inside a list, inside a dictionary:

```
{'Meters': [('item', {'Division': '51', 'Installation': '3000194370', 'Register': '001', 'Equipment': '000000000040099113', 'Oldreading': '29165', 'Newreading': None, 'Meternumber': '00000000008000104'}), ('item', {'Division': '52', 'Installation': '3000417096', 'Register': '001', 'Equipment': '000000000010117702', 'Oldreading': '5846', 'Newreading': None, 'Meternumber': '000000080823161079'})], 'Blocked': None}
```

6.5 Accessing Data in Dictionaries

Elements of a dictionary can be accessed by key in three different ways:

- mydict["somekey"] Throws a KeyError exception if mydict doesn't contain *somekey*
- mydict.get("somekey") Returns None if mydict doesn't contain *somekey*

- mydict.get("somekey",1) Returns a default value (1 in this case) if mydict doesn't contain *somekey*

To prevent a KeyError exception being thrown in the IVR, best practice is to use mydict.get("somekey"). For example, if the dictionary is: mydict = {'Multi': None, 'Meters': None, 'Blocked': 'X'}

mydict["Blocked"]	Returns X
mydict.get("Blocked")	Returns X
mydict["Single“]	Throws a KeyError exception
mydict.get("Single")	Returns None

You can use a variable as the key: myvar = "Blocked"

mydict.get(myvar)	Returns X

6.6 Accessing Data in Dictionaries – Complex Example

Elements of a dictionary within a dictionary can be accessed in the following ways:

```
mydict = {'Return': {'Type': 'E', 'Number': '016', 'Id': 'ZIVRMES‘},
'Multi': None, 'Blocked': 'X'}
```

mydict["Return"]["Type"]	Returns E
mydict.get("Return",{}).get("Type")	Returns E
mydict["Return"]["Letter"]	Throws a KeyError exception
mydict.get("Return",{}).get("Letter")	Returns None
mydict["Result"]["Letter"]	Throws a KeyError exception
mydict.get("Result",{}).get("Letter")	Returns None

Again, in order to prevent a KeyError exception being thrown in the IVR, best practice is to use mydict.get("somekey1",{}).get("somekey2")
The first .get *has* to return a dictionary object in order to avoid a KeyError, so it returns a new empty one (default value) in case there is no *Result* element available in mydict

6.7 Accessing Data in Lists and Tuples

Elements of a list or tuple can only be accessed using a zero-based index. There is no `.get` method available for lists and tuples.

```
mylist = ['foo','bar']
```

mylist[0]	Returns 'foo'
mylist[1]	Returns 'bar'
mylist[-1]	Returns the last element in the list; 'bar' in this case
mylist[2]	Throws an IndexError exception (list index out of range)

Use mylist.count("somevalue") to test if *somevalue* exists. 0 is returned if it doesn’t exist.

mylist.count(“foo")	Returns 1

mylist.count(“food")	Returns 0
mylist[mylist.index('foo')]	Returns ‘foo’ but can be used only if ‘foo’ exists in the list or tuple
len(mylist)	Returns 2 – the number of items in the list

You can append items to a list. For example, mylist + [‘gaa’] results in mylist = ['foo','bar‘,’gaa’]

7 Python Customizer Example

Note: For Sinch Contact Pro cloud, Sinch needs to verify that any customizer you want to add won't cause security or performance issues. Create a service request via the support portal for the customizer review. This is billable work and based on actual review hours.

Python customizers can be used for various purposes with Sinch Contact Center IVR application. This sample describes how to make a database query by using Python customizer. For further details about IVR structure and the sample Python customizer file, contact support. **NOTE:** Custom directory database table is called "*VipId*" in this customizer sample.

Prerequisites:

- Create new table *CustomerID* to the Sinch Contact Center directory database.
- Modify Python customizer to access this table by using Windows authentication.

The customstate element is used to invoke a Python customizer method call and store its results in a variable. For more information about customstate configurations, see the System Configurator (SC) document.

The required values are:

- **(form)** *Identifier* = *FindCustomerFromDatabase* (Parent element for customstate child element)

  This is an identifier (name) for parent element. The identifier appears in the ID column, and it is used as an address when the call is transferred to this element.

- **(customstate)** Identifier = CustomerGUID

  This is an identifier (name) for child element. The identifier appears in the drop-down list of expressions with the prefix customstate:, the results of the customized method are called with that expression.

- **(customstate)** Method Name = getCustomerById

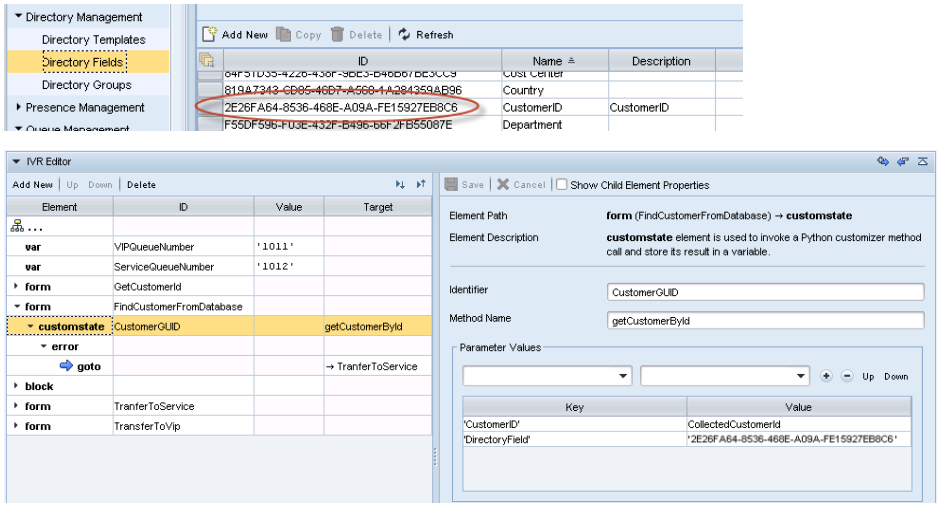
  This specifies the python customer method to call.

- **(customstate)** Parameter Values (Key and Value pairs) = 'CustomerID' and CollectedCustomerId

  - *'CustomerId'* is the name of the custom method parameter. The value of this parameter contains the customer id that is used to search for a customer form the database.
  - *CollectedCustomerId* is an identifier of the field element where the given customer ID DTMF digits are collected.

- **(customstate)** *Parameter Values* (Key and Value pairs) = *'DirectoryField'* and *'GUID'*

  - *'DirectoryField'* is the name of the custom variable for connecting the custom directory field to the appropriate GUID for search purposes in Python customizer.
  - *GUID* is the GUID of the custom directory field in Sinch Contact Center directory database.



7.1 DB Query Customizer Sample

DB Query Customizer provides a good example with comments on how to connect to internal or external databases by using Sinch Contact Center's own ODBC API and use the results. For the customizer, contact support.

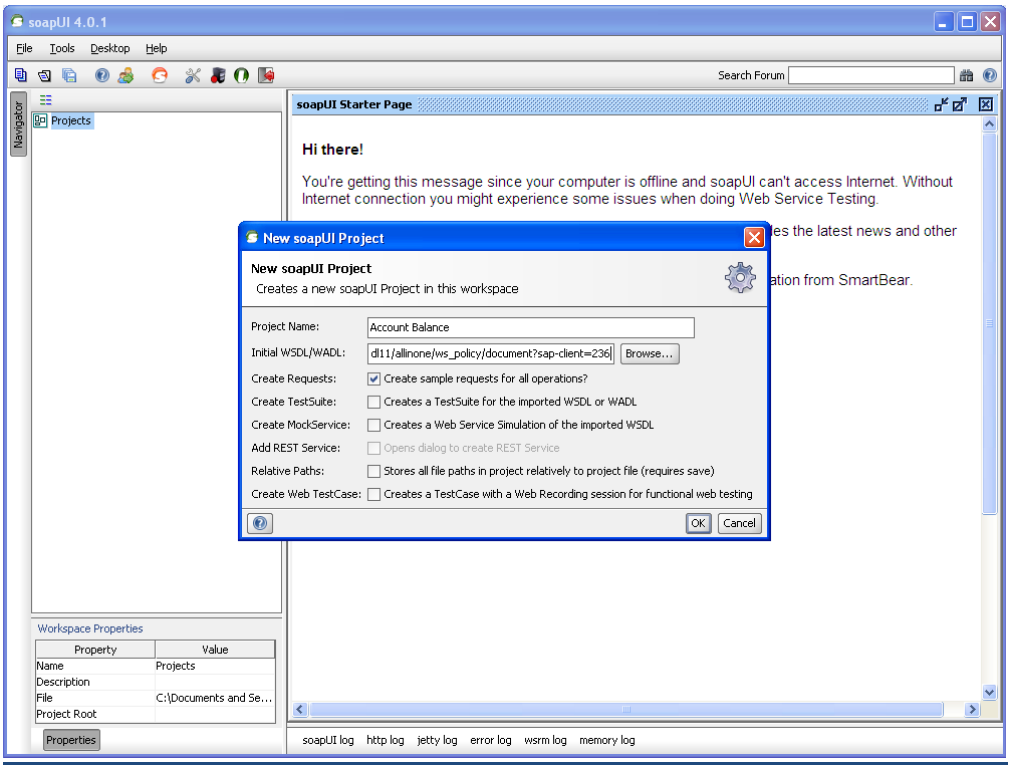
8 SoapUI

Note: In Sinch Contact Pro cloud environments, using SOAP requires making a service request to Sinch support.

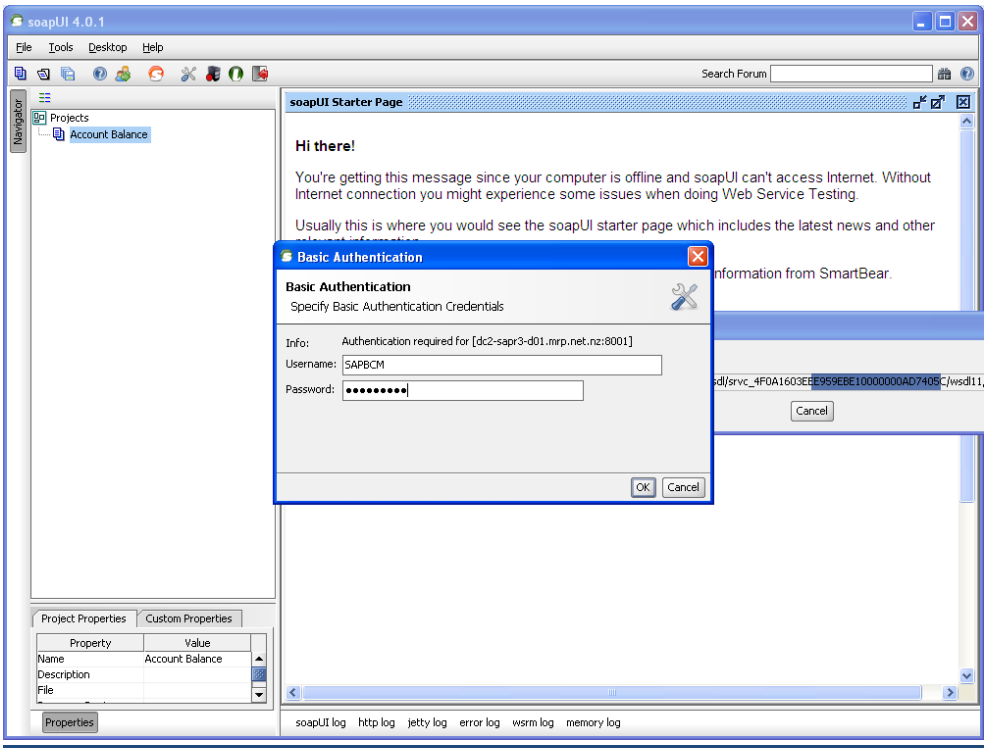
SoapUI can be used to test your web services and to show the structure of the input and output messages. By testing your web services first in SoapUI, you will eliminate errors resulting from the Soap query first, which will make it much easier to use your queries inside your IVR. By installing and running SoapUI from the server(s) where the Sinch Contact Center Core VU is installed, you can first test if there are any firewall issues. SoapUI is freeware and is available from: http://www.soapui.org/

8.1 Test Your Web Services

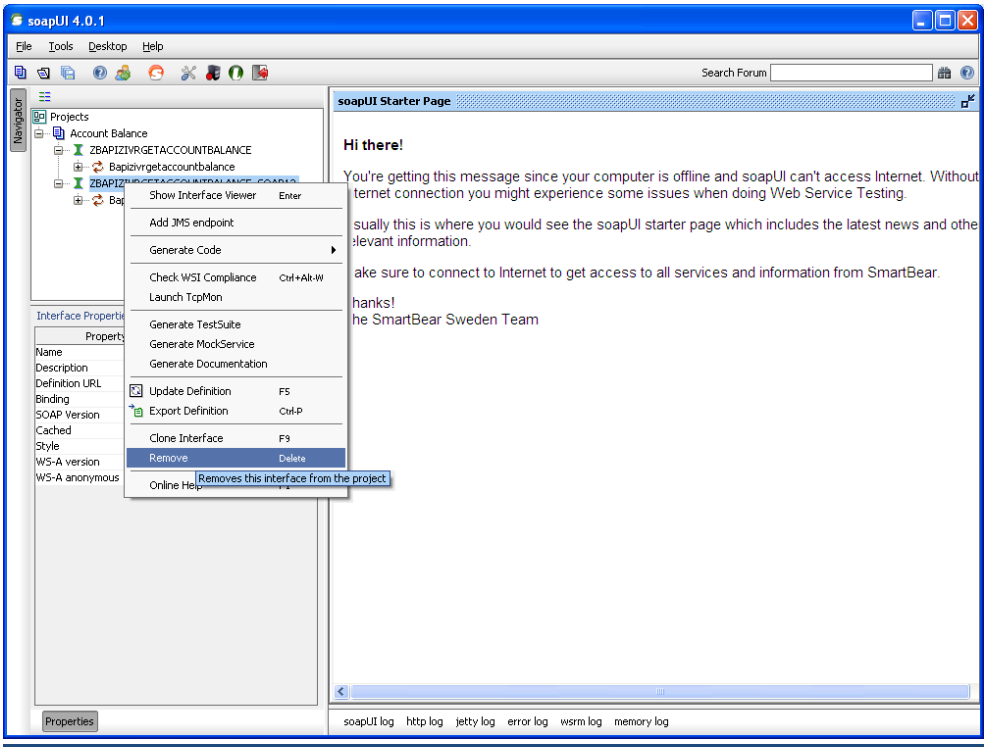
1. Create a new project and insert your wsdl url.



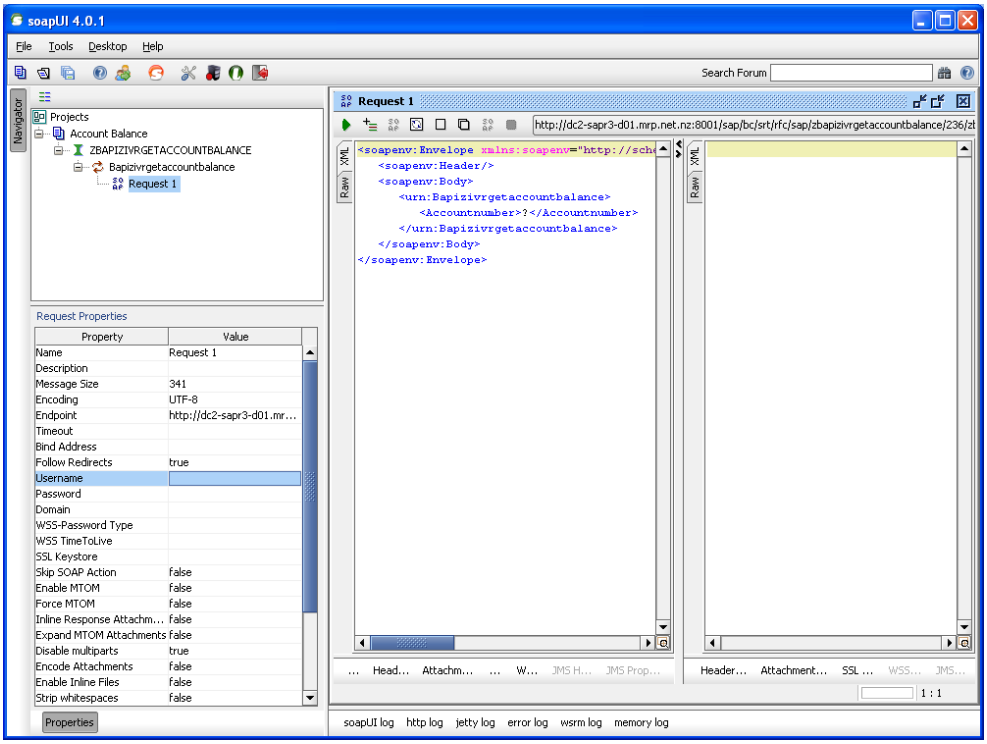
2. Enter the user and password that will be used with the web services (this assumes the web service is set up to use Basic Authentication).



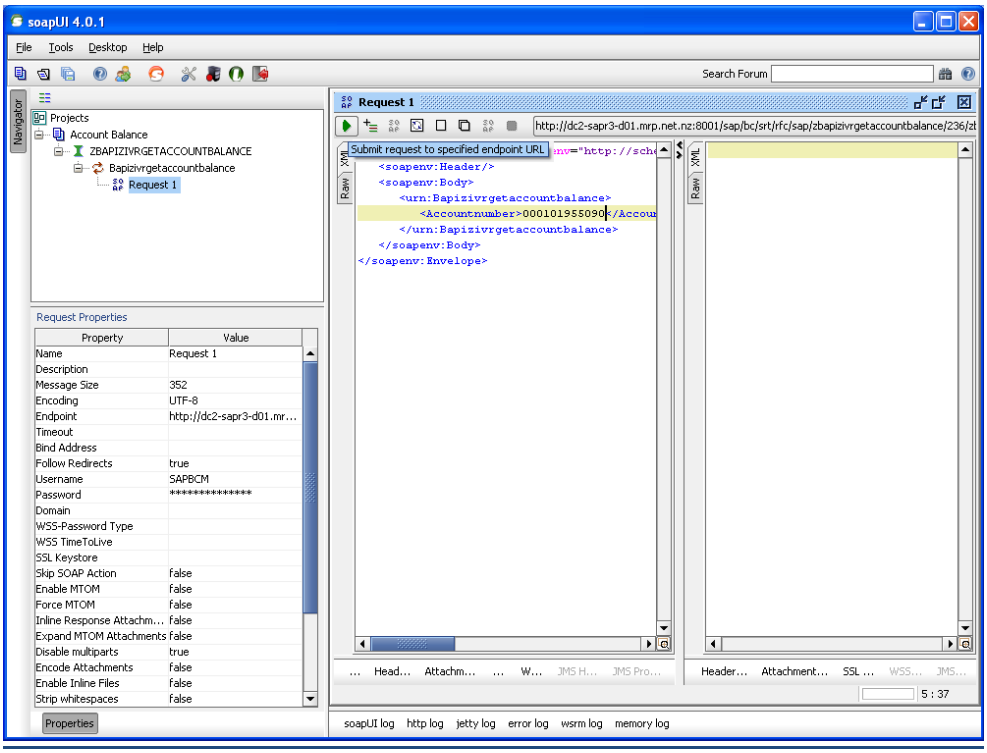
3. Remove the SOAP1.2 interface, as it is not required.



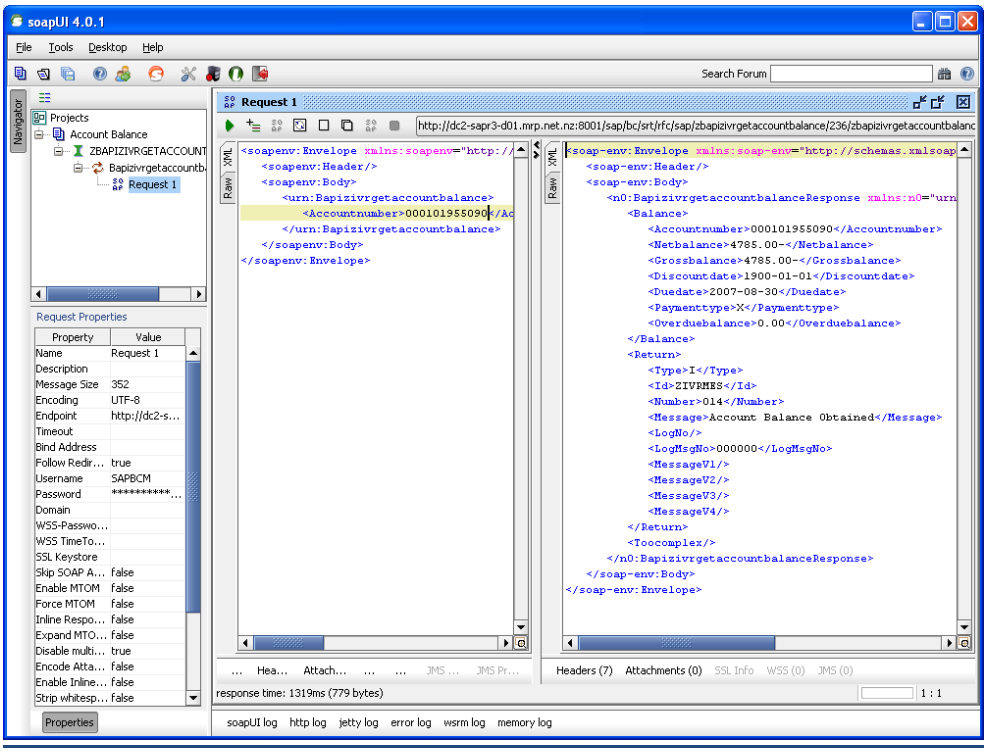
4. Open *Request 1* and enter the user and password that will be used with the web services (this assumes the web service is set up to use Basic Authentication). This is your query structure. We are only interested in the information in the Body.



5. Fill in the parameter and submit the request.



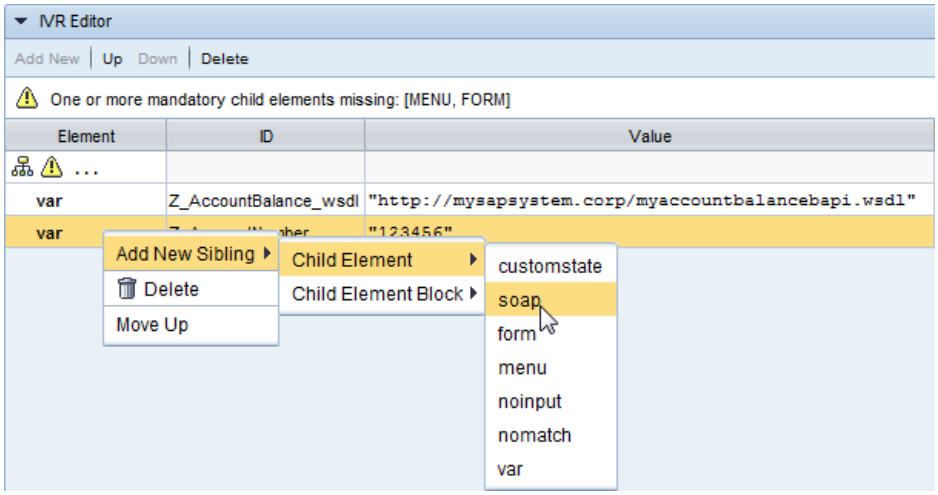
Now you should have your return structure. Again, we are only interested in the information contained in the Body. Running this query in the IVR returns the same structure but in a Python format using dictionaries, lists, and tuples.



8.2 Using Soap Elements inside Sinch Contact Center IVR

The example used in this chapter:

```
<urn:Bapizivrgetaccountbalance>
    <Accountnumber>123456</Accountnumber>
</urn:Bapizivrgetaccountbalance>
```

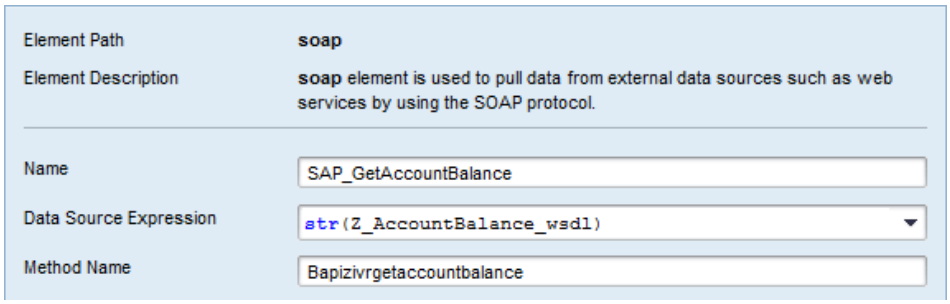


1. Give your Soap element a unique name.
2. Enter the url of your wsdl file between quotes (that is, as a string) or use a variable defined earlier:

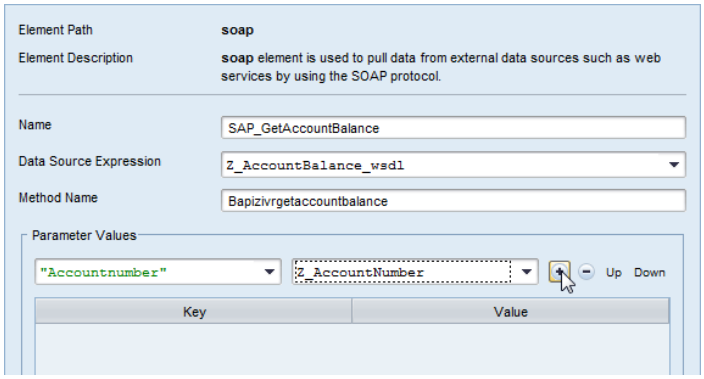
- "http://mysapsystem.corp/myaccountbalancebapi.wsdl"
- str(Z_AccountBalance_wsdl)

3. The method name is the name specified in the urn:

```
<urn:Bapizivrgetaccountbalance>
    <Accountnumber>123456</Accountnumber>
</urn:Bapizivrgetaccountbalance>
```



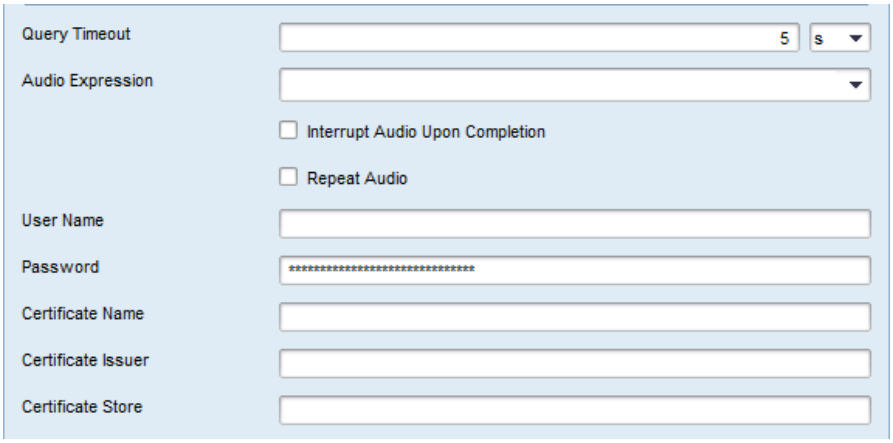
4. Add the input parameters your soap service is expecting: xml child nodes of the urn node.



- The key is the name of the node.
- The key needs to be within quotes to stop it being evaluated as a Python variable. "Accountnumber"
- The value can be a string or a variable.

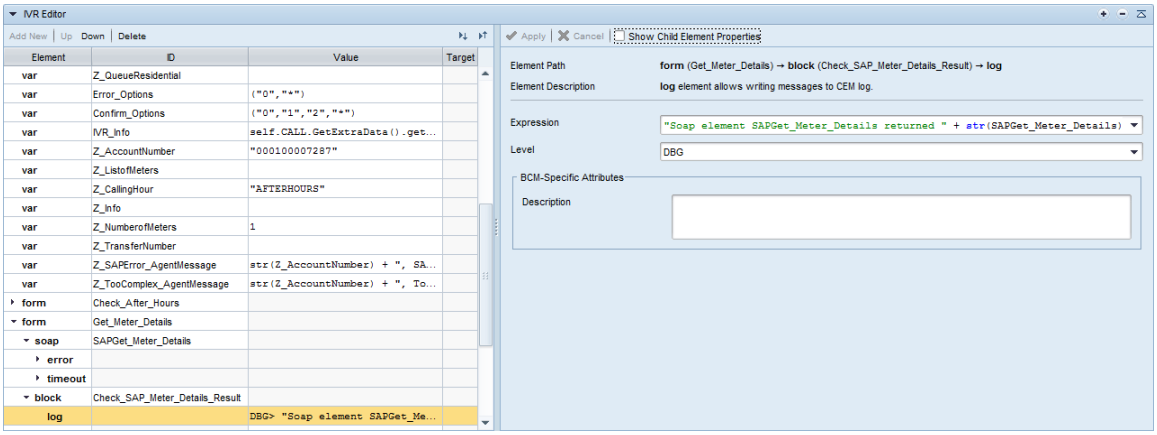
5. Click ⊕ to add your entry.

6. Save your changes so you don't lose your entry.
7. Change the Query Timeout period if required
8. You can assign a variable containing a prompt to be played during execution
9. Enter the credentials as required.
   This depends how your Soap end-point was set up in the target SAP CRM or SAP ERP system in soapmanager.



10. Don't forget to add error and timeout handlers. The system executes the timeout element when the Query Timeout value defined earlier is reached.

8.3 Adding a log entry in your IVR to return a value from a Soap or custom state element



To add a log entry to show the return value from a soap or custom state, add the name of the element in the Expression:

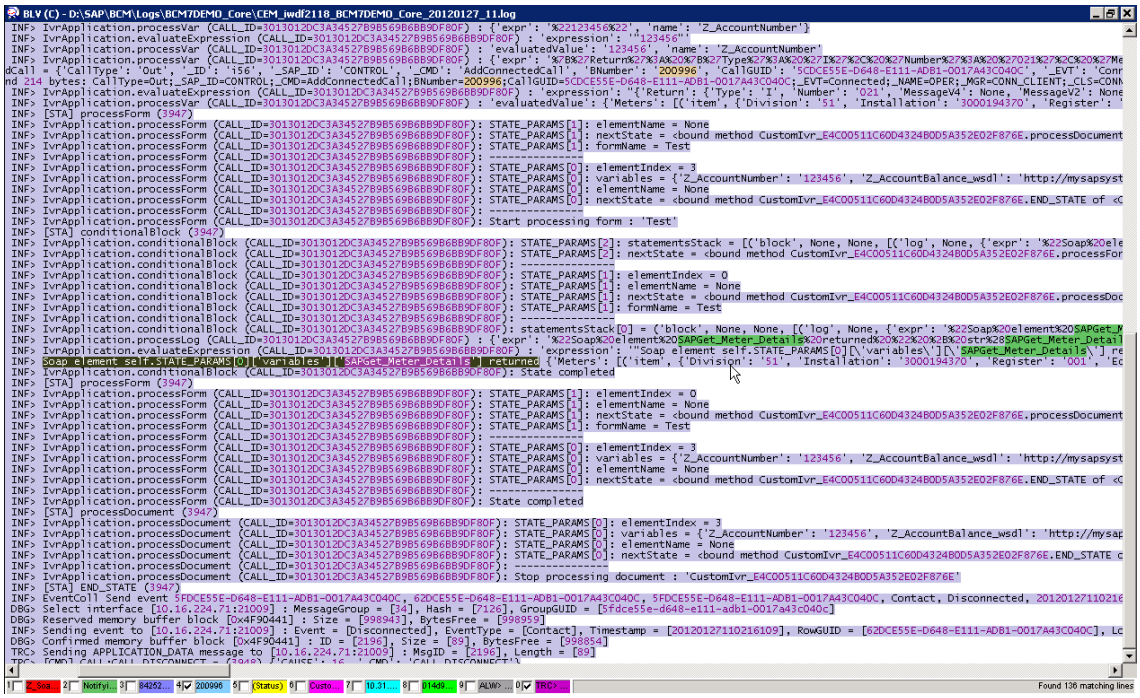
> *Expression = SAPGet_Meter_Details*

By adding a unique text to the log entry, you can find this entry in the CEM log more easily:

> *Expression = " Soap element SAPGet_Meter_Details returned " + str(SAPGet_Meter_Details)*

You can then work out how to parse the result.

Search Result - return value from a soap or custom state element



8.4 Evaluating the results of a Soap query

In the CEM log, you will see the result of the Soap query when you run the IVR.

Examples:

{'Country': BE, 'Number': 475334500, 'Return': X, 'ReturnMsg': None}

{'Toocomplex': None, 'Balance': {'Accountnumber': '000101955090', 'Overduebalance': '0.00', 'Grossbalance': '4785.00-', 'Paymenttype': 'D', 'Netbalance': '4785.00-', 'Discountdate': '1900-01-01', 'Duedate': '2007-08-30'}, 'Return': {'Type': 'I', 'Number': '014', 'MessageV4': None, 'MessageV2': None, 'MessageV3': None, 'Message': 'Account Balance Obtained', 'MessageV1': None, 'LogMsgNo': '000000', 'Id': 'ZIVRMES', 'LogNo': None}}

{'Return': {'Type': 'I', 'Number': '021', 'MessageV4': None, 'MessageV2': None, 'MessageV3': None, 'Message': 'Meters Returned', 'MessageV1': None, 'LogMsgNo': '000000', 'Id': 'ZIVRMES', 'LogNo': None}, 'Multi': None, 'Meters': [('item', {'Division': '51', 'Installation': '3000194370', 'Register': '001', 'Equipment': '000000000040099113', 'Oldreading': '29165', 'Newreading': None, 'Meternumber': '000000000008000104'}), ('item', {'Division': '52', 'Installation': '3000417096', 'Register': '001', 'Equipment': '000000000010117702', 'Oldreading': '5846', 'Newreading': None, 'Meternumber': '00000080823161079'})], 'Blocked': None}

8.5 Using the Python syntax of Dictionaries, Lists and Tuples

The following example is a simple python dictionary with 4 key-value pairs. **Note:** BE and X are not strings but can be var or field.

```
{'Country': BE, 'Number': 475334500, 'Return': X, 'ReturnMsg': None}
```

The following example contains 2 dictionaries: *Balance* and *Return* inside a dictionary with 3 key-value pairs: *Toocomplex*, *Balance* and *Return*.

```
{'Toocomplex': None, 'Balance': {'Accountnumber': '000101955090', 'Overduebalance': '0.00', 'Grossbalance': '4785.00-', 'Paymenttype': 'D', 'Netbalance': '4785.00-', 'Discountdate': '1900-01-01', 'Duedate': '2007-08-30'}, 'Return': {'Type': 'I', 'Number': '014', 'MessageV4': None, 'MessageV2': None, 'MessageV3': None, 'Message': 'Account Balance Obtained', 'MessageV1': None, 'LogMsgNo': '000000', 'Id': 'ZIVRMES', 'LogNo': None}}
```

Assuming your Soap element is called *mysoapquery:*

- To access the value of Toocomplex, you can use: mysoapquery.get("Toocomplex")
- To access the value of Type, you can use: mysoapquery.get("Return",{}).get("Type")

Complex Return structure

```
{'Return': {'Type': 'I', 'Number': '021', 'MessageV4': None, 'MessageV2': None, 'MessageV3': None, 'Message': 'Meters
```

Returned', 'MessageV1': None, 'LogMsgNo': '000000', 'Id': 'ZIVRMES', 'LogNo': None}, 'Multi': None, 'Meters': [('item', {'Division': '51', 'Installation': '3000194370', 'Register': '001', 'Equipment': '000000000040099113', 'Oldreading': '29165', 'Newreading': None, 'Meternumber': '00000000008000104'}), ('item', {'Division': '52', 'Installation': '3000417096', 'Register': '001', 'Equipment': '000000000010117702', 'Oldreading': '5846', 'Newreading': None, 'Meternumber': '00000080823161079'})], 'Blocked': None}

This can be broken down using *mysoapquery.get("Return")* into:

{'Type': 'I', 'Number': '021', 'MessageV4': None, 'MessageV2': None, 'MessageV3': None, 'Message': 'Meters Returned', 'MessageV1': None, 'LogMsgNo': '000000', 'Id': 'ZIVRMES', 'LogNo': None}

Using mysoapquery.get("Meters",{})[0] results in:

('item', {'Division': '51', 'Installation': '3000194370', 'Register': '001', 'Equipment': '000000000040099113', 'Oldreading': '29165', 'Newreading': None, 'Meternumber': '00000000008000104'})

Using mysoapquery.get("Meters",{})[0][1].get("Division") results in:

51

8.6 Using counters

Using a variable as a counter, you can read each of the entries in *Meters.* Assuming we use the variable *mycounter* and assign it an initial value of 0:

- Using *mysoapquery.get("Meters",{})[mycounter]* results in:

> *('item', {'Division': '51', 'Installation': '3000194370', 'Register': '001', 'Equipment': '000000000040099113', 'Oldreading': '29165', 'Newreading': None, 'Meternumber': '000000000008000104'})*

- Using mysoapquery.get("Meters",{})[mycounter][1].get("Division") results in:

> *51*

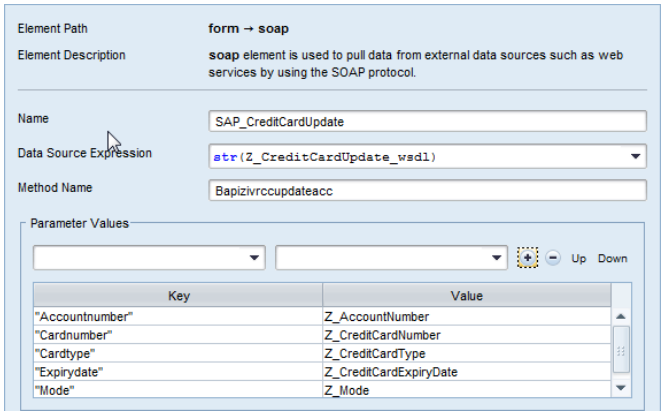
- Incrementing *mycounter* by 1 results in:

> *('item',{'Division': '52', 'Installation': '3000417096', 'Register': '001', 'Equipment': '000000000010117702', 'Oldreading': '5846', 'Newreading': None, 'Meternumber': '000000080823161079'})*
> *52*

8.7 Using Soap elements with multiple input parameters

Add all the input parameters your soap service is expecting: xml child nodes of the urn node.

```
<urn:Bapizivrccupdateacc>
    <Accountnumber>?</Accountnumber>
    <Cardnumber>?</Cardnumber>
    <Cardtype>?</Cardtype>
    <Expirydate>?</Expirydate>
    <Mode>?</Mode>
</urn:Bapizivrccupdateacc>
```



8.8 Complex Query structure – how to add as parameters

8.8.1 Mapping the xml structure to Python elements

The syntax is quite flexible. Some constructs can be expressed in several ways. Others must be expressed with specific syntax.

parameter-value	value
value	simple-value | complex-value
simple-value	number | string | “None”
complex-value	node-dictionary | node-list | node-tuple
node-dictionary	“{“ node-name “:” value [“,”…] “}”
node-list	“(“ complex-value [“,” …] “)”
node-tuple	“(“ node-name [“,” attributes [“,” value]] “)”
attributes	attribute-dictionary | “None”
attribute-dictionary	“{“ attribute-name “:” simple-value [“,” …] “}”
node-name	string
attribute-name	string

8.8.2 Mapping the xml structure to Python syntax

You must use the list syntax for an element with sub-elements if there are:

- Sub-elements with attributes. Attributes are specified as a dictionary within a list:

```
<urn:somerequest>
    <ids>
            <id name="ID" value="Card">123</id>
    </ids>
</urn:somerequest>
```

Key	Value
"ids"	("id",{"name": "ID","value":"Card"},"123")

- Multiple instances of some sub-element. There are no attributes here so the second item in each list is None:

```
<urn:somerequest>
    <ids>
        <id>123</id>
        <id>321</id>
    </ids>
</urn:somerequest>
```

Key	Value
"ids"	(("id",None,"123"),("id",None,"321"))

In other cases, you can use either dictionary or list syntax for an element:

```
<urn:somerequest>
    <person>
        <firstname>John</firstname>
        <surname>Smith</surname>
    </person>
</urn:somerequest>
```

Dictionary syntax:

Key	Value
"person"	{"firstname": "John","surname": "Smith"}

List syntax:

Key	Value
"person"	(("firstname",None,"John"),("surname",None,"Smith"))

Here we give the list of meters as a list of lists. Each list item is then a tuple of three things: node name, attributes, and value. The value is just a dictionary specifying the sub-nodes.

```
<urn:Bapizivrmrupdate>
<Accountnumber>1234567890</Accountnumber>
<Meters>
    <item>
        <Installation>Inst A</Installation>
```

```
            <Meternumber>Meter 1</Meternumber>
            <Newreading>12345</Newreading>
        </item>
        <item>
            <Installation>Inst B</Installation>
            <Meternumber>Meter B</Meternumber>
            <Newreading>54321</Newreading>
        </item>
</Meters>
</urn:Bapizivrmrupdate>
```

Key	Value
"Accountnumber"	1234567890
"Meters"	(("Item",None,{"Installation":"Inst A","Meternumber":"Meter 1","Newreading":12345}),("Item",None,{"Installation":"Inst B","Meternumber":"Meter 2","Newreading":54321}))

9 Appendix 2

Agent presence can be revealed with the `GetNumberInfo` function:

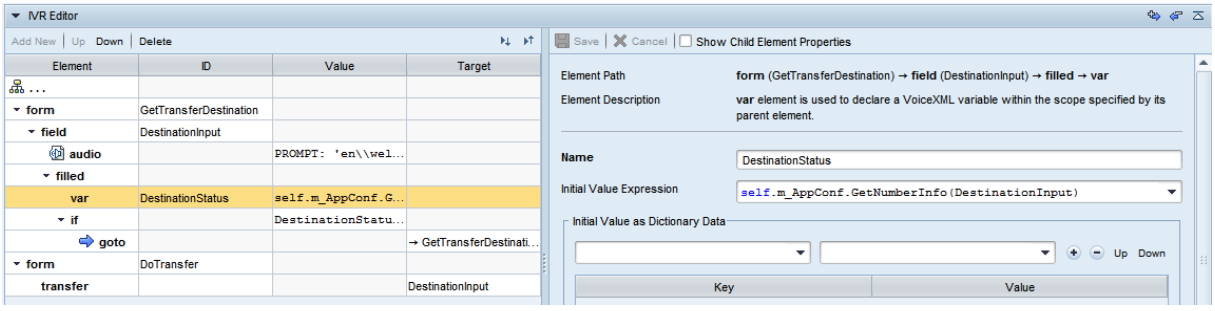
```
self.m_AppConf.GetNumberInfo('<agent extension number>')
```

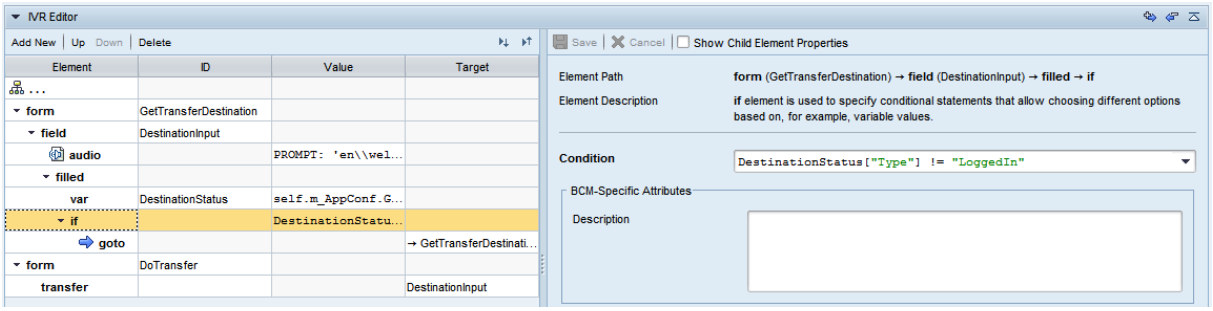
The function returns a Python dictionary containing information of the user.

Examples of return values:

- **Wrong Number:**
  `{'Type': 'UNKNOWN', 'OrgNbr': '67676767', 'MappedNbr': '67676767'}`
- **Agent logged out:**
  `{'IvrLanguage': 'ET', 'MappedNbr': '4401', 'Language': 'EN', 'MaxQue': 3, 'PBX': '', 'OrgNbr': '4401', 'FwdTo': '4555', 'Type': 'LoggedOut'}`
- **Agent logged in and free (Presence-profile, free):**
  `{'Available': 1, 'IvrLanguage': 'ET', 'Slave': 0, 'MappedNbr': '4401', 'Language': 'EN', 'Calls': 0, 'UIStatus': 'StatusPaperWork', 'MaxQue': 3, 'PBX': '', 'Paused': 0, 'OrgNbr': '4401', 'FwdTo': '4555', 'Outbound': 0, 'Type': 'LoggedIn'}`
- **Agent logged in and busy (Presence-profile, busy):**
  `{'Available': 0, 'IvrLanguage': 'ET', 'Slave': 0, 'MappedNbr': '4401', 'Language': 'EN', 'Calls': 1, 'UIStatus': 'StatusPaperWork', 'MaxQue': 3, 'PBX': '', 'Paused': 0, 'OrgNbr': '4401', 'FwdTo': '4555', 'Outbound': 0, 'Type': 'LoggedIn'}`
- **Agent logged in and free (Absence-profile, free)**
  `{'Available': 0, 'IvrLanguage': 'ET', 'Slave': 0, 'MappedNbr': '4401', 'Language': 'EN', 'Calls': 0, 'UIStatus': 'StatusPause', 'MaxQue': 3, 'PBX': '', 'Paused': 1, 'OrgNbr': '4401', 'FwdTo': '4555', 'Outbound': 0, 'Type': 'LoggedIn'}`

The function can be called and used in the following way:





10 Appendix 3

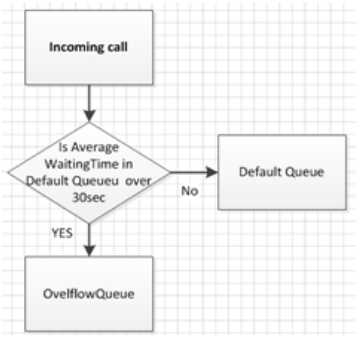
QUEUE.QUERY example 1

The task is to create a conditional IVR that forwards calls to a secondary queue if average waiting time in the primary queue exceeds a specified time value. We modify the ready-made example that checks if there are more calls waiting than the defined maximum number of waiting calls, so that the average waiting time is used as a critical factor.
Also, instead of a customer parameter recommended in the original example, a variable is added.

Primary Queue = 2100 = DefaultQueue
Secondary Queue = 2101 = OverflowQueue
AverageWaiting = 5 = time in seconds

Steps to create:

1. Import the “Example_IVR_Conditional.xml” from the Examples and Templates folder.
2. Add Element <var> ID “AverageWaiting” Value “5” and <var> ID “OverflowQueue” Value”2101”
3. Modify DefaultQueue value to be “2100”
4. Modify Element <if> by selecting from the dropdown menu value “Queue Status Query: Average Waiting Time” and edit the line to be like QUEUE.QUERY(str(DefaultQueue), “Sta_AverageWaitingTime”) > int(AverageWaiting)
5. Modify Element <assign> Value to point to “OverflowQueue”
6. Modify attribute Descriptions accordingly as they are shown in the log files.



IVR Editor

Add New | Up Down | Delete

Element	ID	Value	Target
...			
var	DefaultQueue	2100	
var	OverflowQueue	2101	
var	DestinationQueue	DefaultQueue	
var	AverageWaiting	5	
form	F1		
block			
if		QUEUE.QUERY(str(...	
assign		OverflowQueue	DestinationQueue
form	F2		
transfer			DestinationQueue

11 Appendix 4

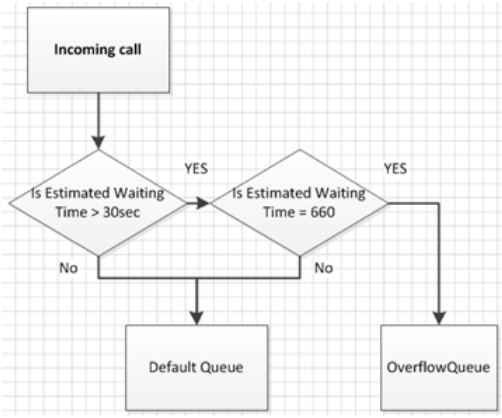
QUEUE.QUERY sample 2

Task is to create an conditional IVR that forwards calls to secondary queue if estimated waiting time in primary queue exceeds spesified time value. To verify functionality add counters to log printing.
Primary Queue = 2100 = DefaultQueue
Secondary Queue = 2101 = OverflowQueue
EstWaiting = 30 = Estimated Waiting Time in seconds
QueMaxWaitTime = 660 = Default value if not defined in SC “Queue Management”> “Max.Waiting Time”

Steps to create:

1. Import the “Example_IVR_Conditional.xml” from the Examples and Templates folder.
2. Add Element <var> ID “AverageWaiting” Value “5” and <var> ID “OverflowQueue” Value”2101”
3. Add <var> EstWaiting, Value 30 and <var>QueMAxWaitTime, Value 660 and Modify DefaultQueue value to be “2100”
4. Modify Element <if> by selecting from the dropdown menu value “Queue Query: Estimated Waiting Time” and edit the line to be like QUEUE.QUERY(str(DefaultQueue), "Que_EstWaitTime") > int(EstWaiting)
5. Add Element <if> from the dropdown menu value “Queue Query: Estimated Waiting Time”: QUEUE.QUERY(str(DefaultQueue), "Que_EstWaitTime") < int(QueMaxWaitTime)
6. Modify Element <assign> Value to point to “OverflowQueue”
7. Add New Child Element <log> by selecting froom dropdown and edit “Expression”:
8. "IvrApplication.logprint (CALL_ID= " + str({CALLID}) + " Average waiting time = " + str(QUEUE.QUERY(str(DefaultQueue), "Sta_AverageWaitingTime"))

9. Add <log>with expression: "IvrApplication.logprint (CALL_ID= " + str({CALLID}) + " Got over estwait time limit, FWD to OverflowQue, EstWait= " + str(QUEUE.QUERY(str(DefaultQueue), "Que_EstWaitTime"))
10. Modify attribute Descriptions accordingly as they are shown in the log files.



IVR Editor

Add New | Up | Down | Delete

Element	ID	Value	Target
...			
var	EstWaiting	30	
var	QueMaxWaitTime	660	
var	DefaultQueue	2100	
var	OverflowQueue	2101	
var	DestinationQueue	DefaultQueue	
form	F1		
block			
if		QUEUE.QUERY(str(Defa...	
if		QUEUE.QUERY(str(Defa...	
log		DBG> "IvrApplication...	
assign		OverflowQueue	DestinationQueue
log		DBG> "IvrApplication...	
form	F2		
transfer			DestinationQueue