Mitel MiContact Center Enterprise

BASIC CALL HANDLING AND ADVANCED APPLICATIONS SCRIPT MANAGER - USER GUIDE

Release 9.5 SP2

NOTICE

The information contained in this document is believed to be accurate in all respects but is not warranted by Mitel Networks ™ Corporation (MITEL®). The information is subject to change without notice and should not be construed in any way as a commitment by Mitel or any of its affiliates or subsidiaries. Mitel and its affiliates and subsidiaries assume no responsibility for any errors or omissions in this document. Revisions of this document or new editions of it may be issued to incorporate such changes.

TRADEMARKS

Basic Call Handling and Advanced Applications Script Manager
User Guide
Release 9.5 SP2 – October 2021

INTRODUCTION

This document contains tutorials for a simple call handling script and more advanced system components. The tutorials help with building powerful and complex scripts in a well-organized way.

The following are described:

- Building a simple script to handle an incoming voice call
- Building an advanced script using sub-scripts and main-scripts.
- Building an advanced script by triggering the session at start up
- How to use the System Event and System Event handler
- How to use the User Exception and User Exception handler
- How to use the User Event and Global Key
- Using the JScriptExecute
- Using the VBScriptExecute

BEFORE USING THE TUTORIALS

Make sure the sample scripts are installed on your development PC. Refer to Installing Sample Scripts, for installation instructions.

Note: For Subscripts an ODBC compliant database is required; see *Database Component Applications*.

INCOMING CALL HANDLER

The incoming call handler script (IncomingCallHandler.mfd) demonstrates how an incoming voice call can be handled.

Open the script from Script Designer by opening the project named `tutorials.fdp` from the SampleScripts directory. Double-click on **IncomingCallHandler** to open the script.

This tutorial emphasizes the OnCallDelivered, AllocateResources, MenuSelection and Play components.

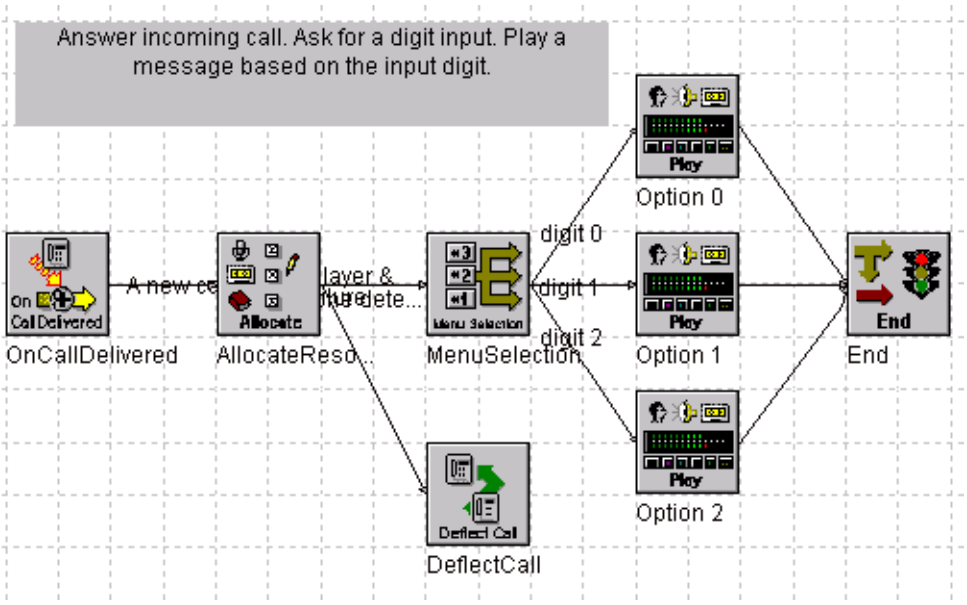


Figure 1: IncomingCallHandler script

The script receives an incoming call and allocates a player and tone detector resource. The caller is prompted to select an option. The script plays a different message based on the user selection and ends after the message is played.

The first block in the Event-Driven section is the OnCallDelivered block. This OnCallDelivered block specifies the device(s) to be monitored in this script.

ONCALL DELIVERED

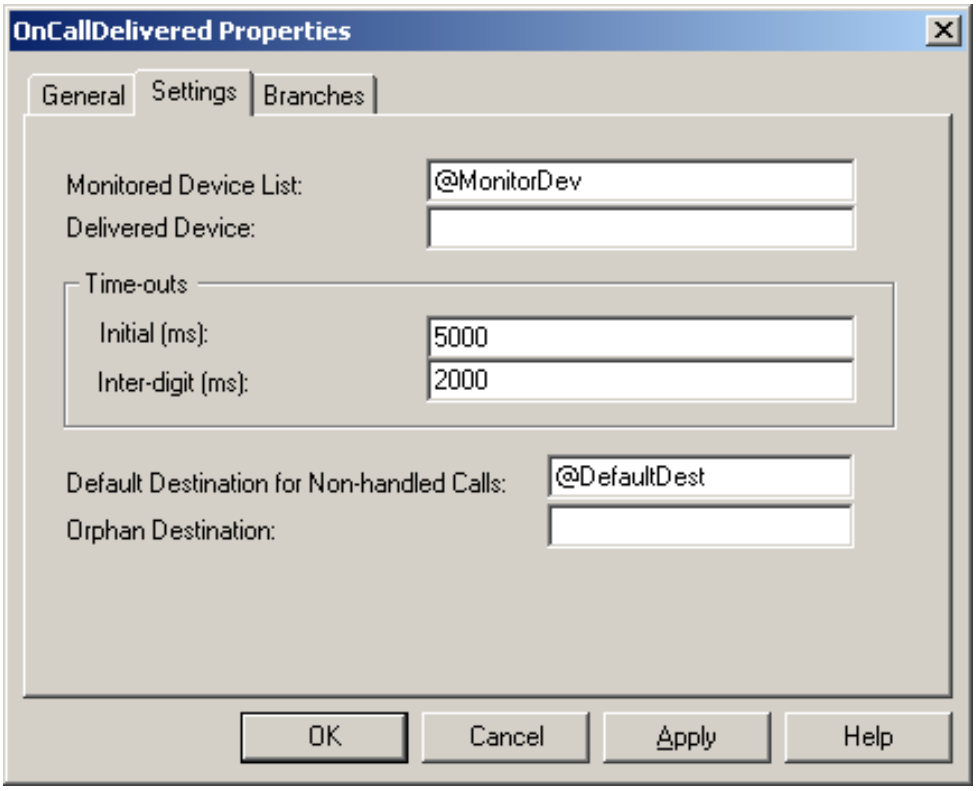


Figure 2: Settings tab of OnCallDelivered Properties dialog box

Monitored Device List

The monitored device is a Basic Virtual Device (BVD) defined in the OAS server. Incoming calls are received via this device. One or more devices can be entered in the Monitored Device List input field. Separate each device name by comma or semicolon.

For a single OAS server configuration, enter one or more BVD names. For example:

- SalesBVD
- MarketingBVD
- SalesBVD, MarketingBVD

For a multiple OAS server configuration, enter one or more BVD names prefixed with the OAS server name. The format is the OAS server name followed by a colon, then the basic virtual device name. For example:

- OAS-LA:SalesBVD
- OAS-LA:SalesBVD, OAS-NY:MarketingBVD,
- OAS-TX:SupportBVD

- OAS-LA:SalesBVD;@MonitorDevice1
  @MonitorDevice1 is a string variable with the initial value of OAS-NY:MarketingBVD, or it can be a configured variable which will be assigned with the value OAS-NY:MarketingBVD.
- @MonitorDevice1, @MonitorDevice2
  @MonitorDevice1 is a string variable. The value can be configured as a script variable.
- @MonitorDevice2 is a string variable. The value can be configured as a script variable.

Note: The OAS server name used must match the defined Call Manager Server in the Site Configuration of MiCC Enterprise Configuration Manager in the Name field. If the Call Manager Server Name is defined as CallMgr1 in the Site Details, the BVD should be defined as CallMgr1:SalesBVD.

Note: The following formats are not valid and will cause the script to fail to start. In both examples the monitor device contains more than one device in a single string or variable:

- "OAS-LA:BVD1,OAS-NY:BVD2"

@MonitorDevice1
Where @MonitorDevice1 has the value "OAS-LA:BVD1,OAS-NY:BVD2"

Delivered Device

The user can supply a string variable that stores the name of the device when a call arrives. This device name can be used to determine if a different path should be taken in the script. This should be created as a session variable since the device could be different for each session.

Default Destination for non-handled calls

In the case that the script takes a branch for which the user did not define a connecting block, the call will be transferred to this destination. The default destination can be a number within or outside the OAS domain. Make sure proper access codes are included when sending a call to another site.

Variables

In this example, configurable variables named "@MonitorDev" and "@DefaultDest" are used. The advantage of using a variable is that it can be modified during service creation time without modifying the script. These variables are defined in the script object pane.

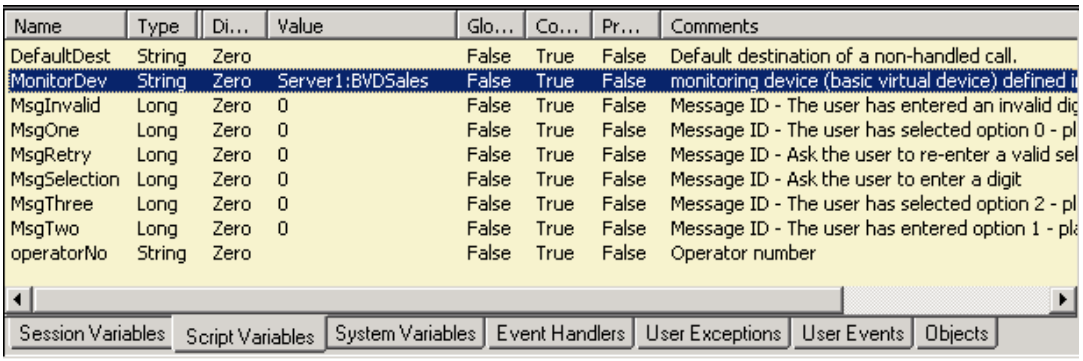


Figure 3: Script Object pane

Double click **MonitorDev** to open the **Variable Information** dialog.

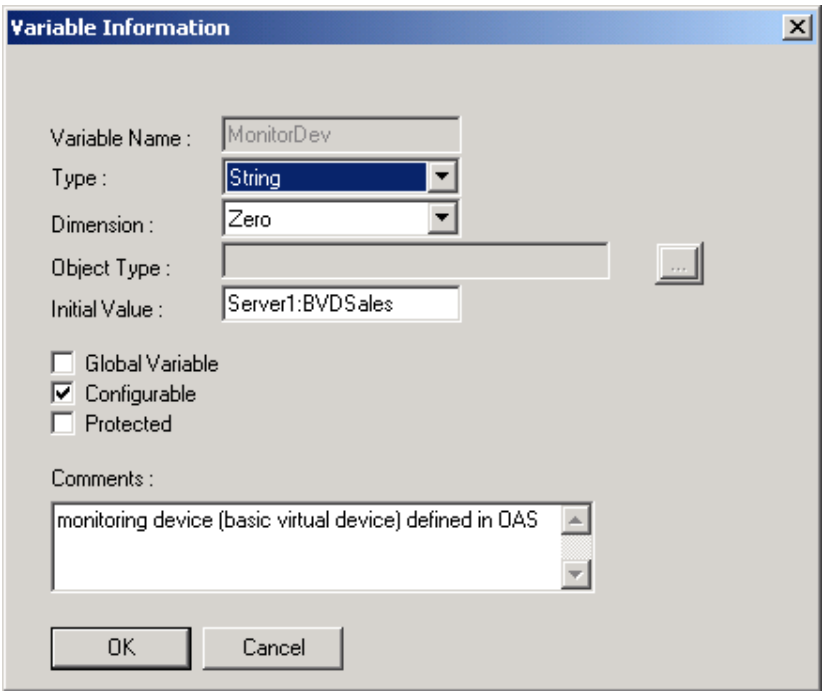


Figure 4: Variable Information dialog box

In this example, the MonitorDev variable is defined as a Script Variable and has an initial value of Server1:BVDSales. The Configurable option must be enabled if this value is to be modified during the Service Application creation time. If the configurable option is not enabled, the variable will not be available for modification via Configuration Manager or Script Manager Configuration. Hence, the value of MonitorDev remains with the initial value or empty. This could cause the activation of the Service Application to fail if a valid BVD name is not configured.

MonitorDev is defined as a Script Variable since it will be used in all sessions for the same Service Application. The value does not change from session to session.

The default destination for non-handled calls is used when the script encounters a selected branch which is not defined. If the call is still active, it will be diverted to the default destination, for example an operator to handle the call. One possible use of this feature would be to perform a customer survey at the end of the call.

After detecting an incoming call, the script allocates the play and tone detection resources to prepare to play a message and collect DTMF digits.

ALLOCATERESOURCES

To allocate resources:

1. Select the settings tab of the AllocateResource Properties dialog box.
2. Select **AllocateResouces and Answer Call** for normal incoming call handling (see Future call section for the usage of other options).
3. Select a **Player** resource to play a voice prompt
4. Select the **Tone Detector** to collect DTMF digits from the caller
5. Mark **Selection** in the Resource Characteristics field, and choose the system prompt language from the drop-down list. Select the Call Manager from the list.
6. When resources are allocated, the success branch will go to the **MenuSelection** block. This block prompts the user to enter a digit.

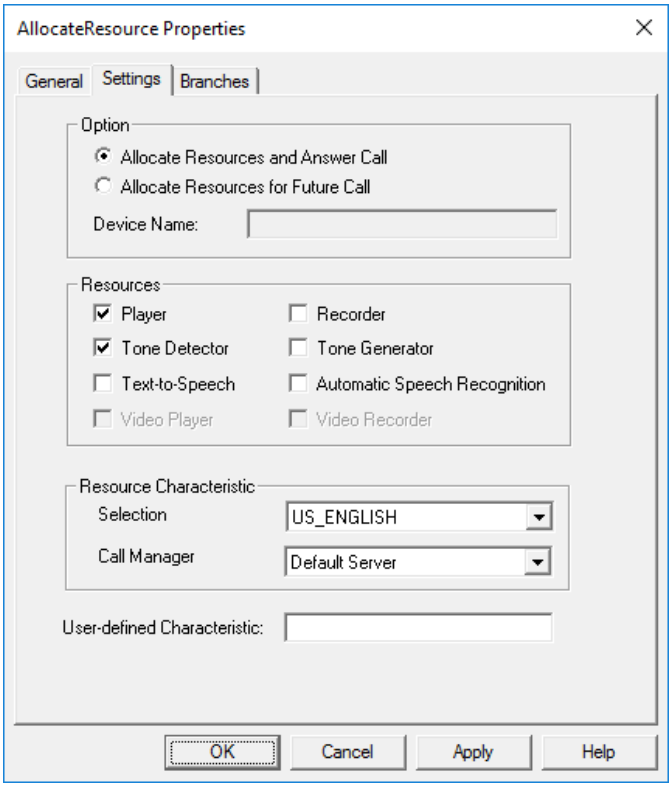


Figure 5: Settings tab of AllocateResource Properties dialog box

MENUSELECTION

This block plays the messages specified in the Messages input fields. It is possible to configure the messages. In the example below, @MsgSelection contains the message ID that will be played in this block.

The **Key Detection Options** change the behavior of the block. If **Flush Pre-entered Keys** is selected, any digit entered before this block will be discarded. The script will start the digit detection when this block is started. This means the user will at least hear a brief message before it is interrupted by the digit entered. If this option is not set and a digit was entered before the block execution, the message will not be played.

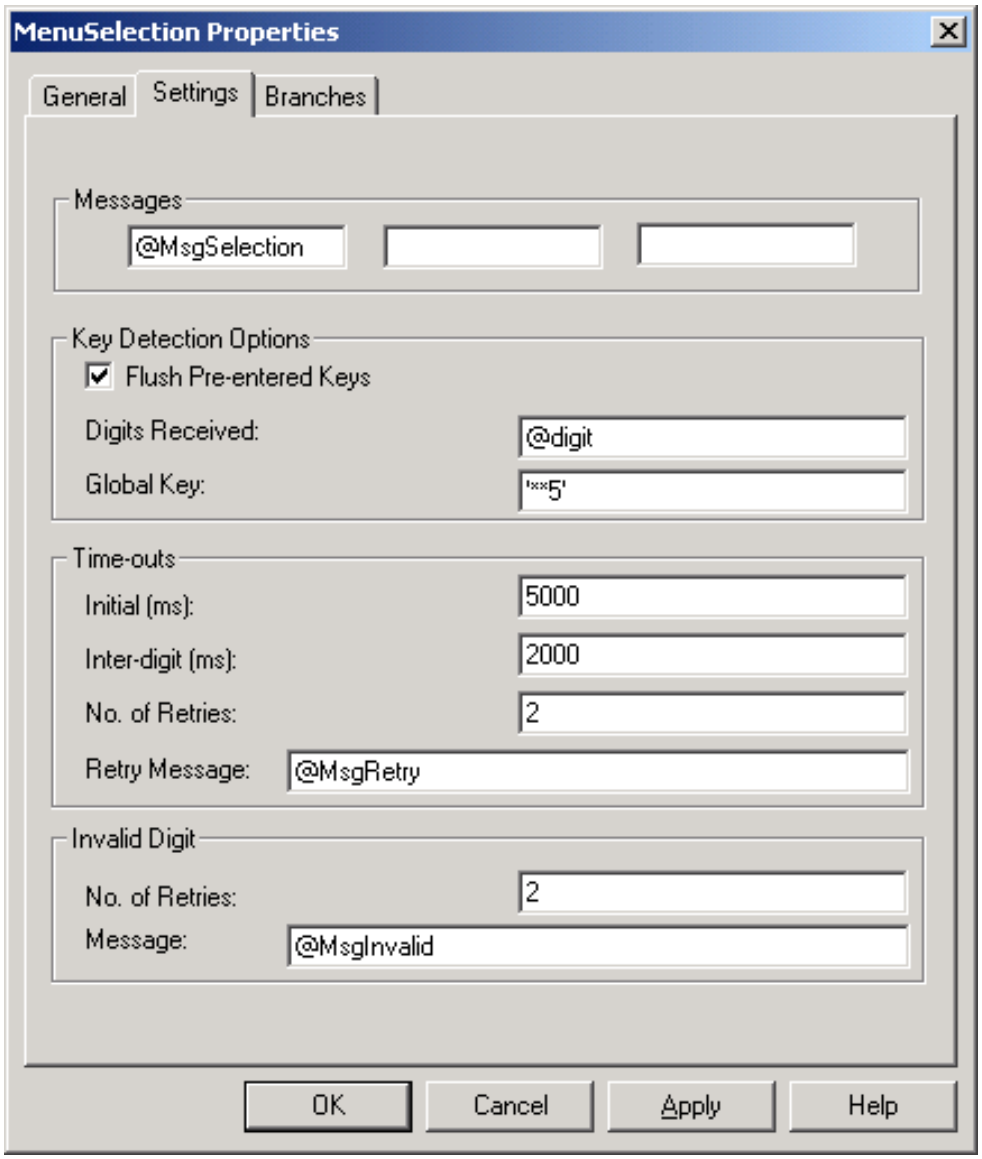


Figure 6: Settings tab of MenuSelection Properties

The digit used in this block will be saved in the @digit variable. Hence the same digit will be used to determine which path will be taken in the next block.

The Global Key is a sequence of keys that can be used to jump to different points in the script. The Global Key takes precedence over the digit selection. For example, if the global key is“**5” and one of the selections is “*”, there will be a delay, based on the inter-digit time-out setting before the system can determine if a selection has been pressed or a global key has been used. If the global key is detected, the block will take the Global Key Detected branch. The global key inter-digit time-out is defaulted to 1 second.

One usage of the Global Key is to jump to a Help Menu. When the Global Key is detected, it can add a block to send a user event. The user event has a user event handling block that plays the help commands. At the end of the command, the script could return to the block that generated the user event. This is another advanced technique to build a complex application.

See User Event and Global Key for more examples.

After the user enters a valid digit, the script continues on the selected branch to play a different message to the caller.

PLAY

The **Play** block, can play up to three messages. These messages can be set to allow or not allow interrupt by digit. If interrupt is enabled, the digit that interrupted the message can be retained for the next **GetDigits** block. An option is also provided to flush digits entered before the block. This governs whether the user will hear at least some part of the voice prompt.

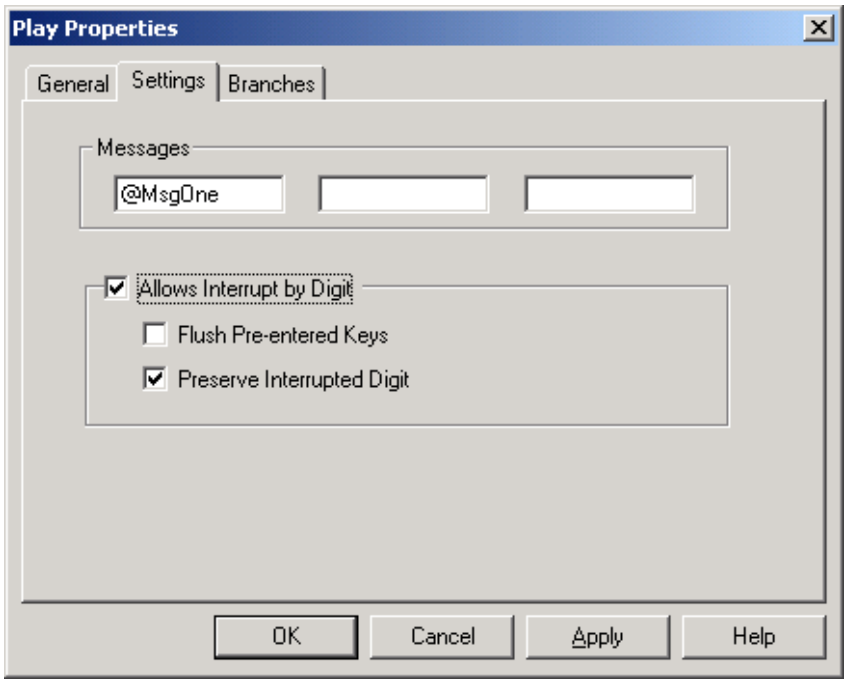


Figure 7: Settings tab of Play Properties

VOICE PROMPTS

The voice prompts, and their default IDs, need to be recorded and configured as play messages in the OAS server. If a different message ID is used in the OAS server, reconfigure the variables with the new message ID when creating the Service Access.

Table 1: Voice Prompts

SCRIPT MANAGER VARIABLE	MESSAGE ID	SAMPLE VOICE PROMPT DESTINATION
MsgInvalid	1200	“You have entered an invalid option”
MsgOne	1201	“You have selected… Thank you for calling.”
MsgRetry	1202	“Please enter your choice of service.”
MsgSelection	1203	“For Sales, press 0. For customer service, press 1…”
MsgThree	1204	“You have selected… Thank you for calling.”
MsgFour	1205	“You have selected… Thank you for calling.”

TESTING THE APPLICATION

To test the application:

1. Create a Service Access in Configuration Manager. It is very important to configure the monitored devices and the play messages; otherwise the Service Application will not operate properly.
2. Click **Variable...** to configure the MonitorDev variable and verify that the other variables are correct.
3. Activate the Service Application and test it.
4. Use SpyTracer to view what is happening in your application in real-time. For more information on SpyTracer, see document *Debugging Applications*.

SUBSCRIPTS

Subscripts are a useful way to reuse frequently used functions, and to separate functionality so that a complex script is more readable and easier to maintain. Subscripts can, for example, be used for validating a PIN code or retrieving user information. This tutorial demonstrates how to use a subscript and how to pass data between the calling script and the subscript. This tutorial requires an ODBC compliant database; see document *Database Components Applications* for details.

If a subscript has been modified by adding or removing input parameters, or has changed its configurable variables, the subscript must be recompiled for the changes to take effect. When subscripts are modified, the calling script must be recompiled.

This tutorial demonstrates two different methods of passing and returning data between subscripts and a calling script, with variable mapping and global variables.

DESIGNING SUBSCRIPTS

Before writing a subscript, identify which commonly used functions can benefit from being written as a subscript. Then define the input and the return data to the subscript.

1. Define the function of a subscript
   Each subscript must have a unique function that the calling script expects it to do. For example, given the caller's phone number, the subscript could return the customer's account number.
2. Define the input and output of the subscript
   Most of the subscripts require some input information. In many cases, the calling script also expects an outcome from the subscript execution, whether it is returned data or just a result.

   There are three ways to pass input data to the subscript: input parameters, variable mapping and global variables.

INPUT PARAMETERS

The user defines input data to the subscript in the **Input Parameters** tab, available in the object pane. The input parameters are the expected input from the calling script. If the calling script does not provide the input data, the value of the data is undefined. In that case, the outcome of the subscript should not be based on the undefined parameter.

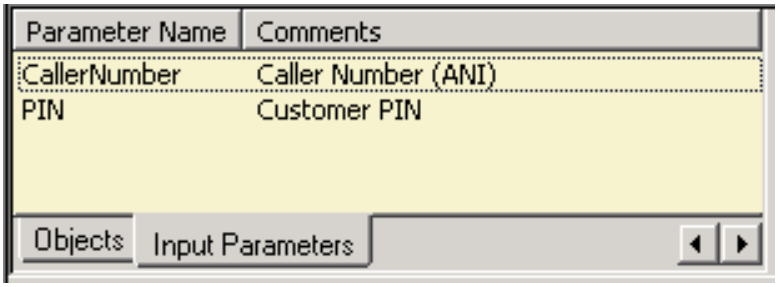


Figure 8: Input Parameters tab

Once the inputs are defined in the subscript, each parameter must map to a variable in the subscript before it can be used. Double-click on the parameter name to open the **Input Parameter mapping** dialog.

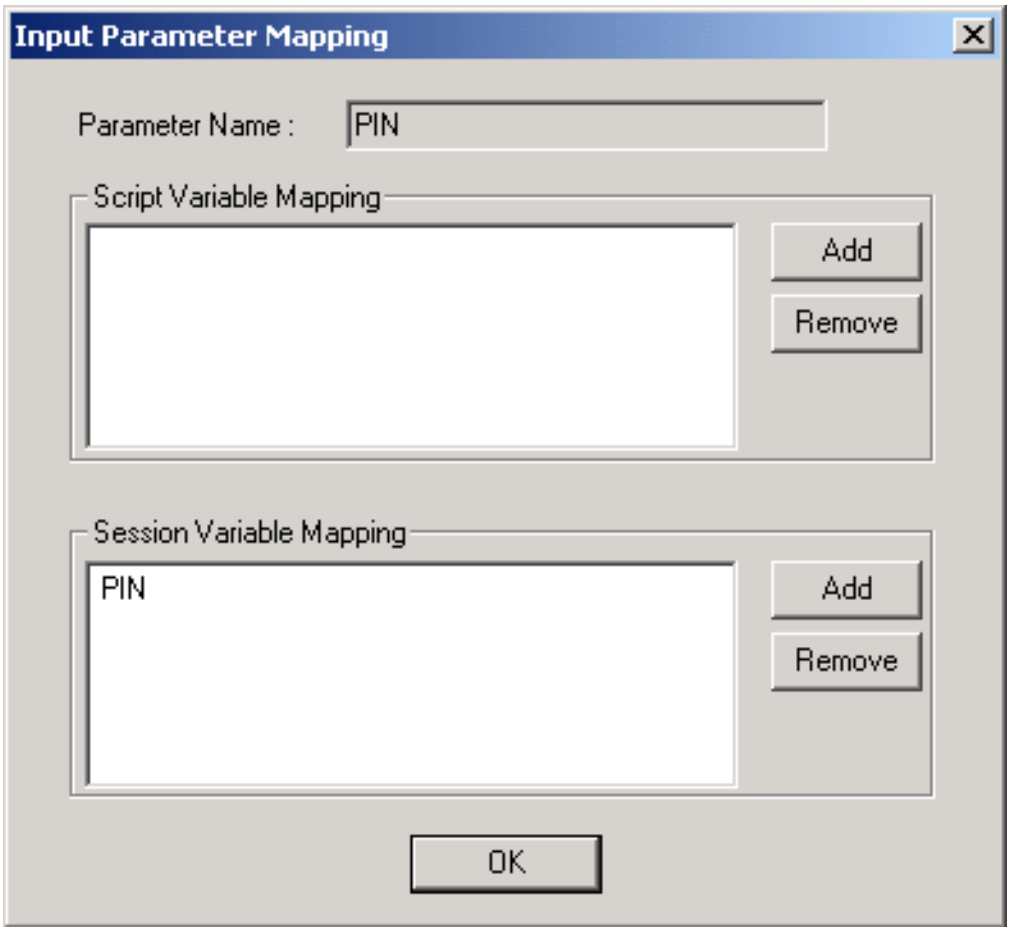


Figure 9: Input Parameter Mapping dialog

The input parameter can be mapped to a subscript session variable or script variable. The input parameter value will be used as the initial value of the mapped variable. Changing the mapped variable in the subscript does not change the value in the calling script.

Calling script settings

The Input Parameters defined in the subscript will be displayed in the **Parameters** tab of the **Sub-script Properties** block of the calling script.

Double-click on the parameter name to open the **Setting Input Parameter Information** dialog. If no input is provided to the subscript, check the **Disabled** button. Otherwise, provide the input data in the **Data** text box.

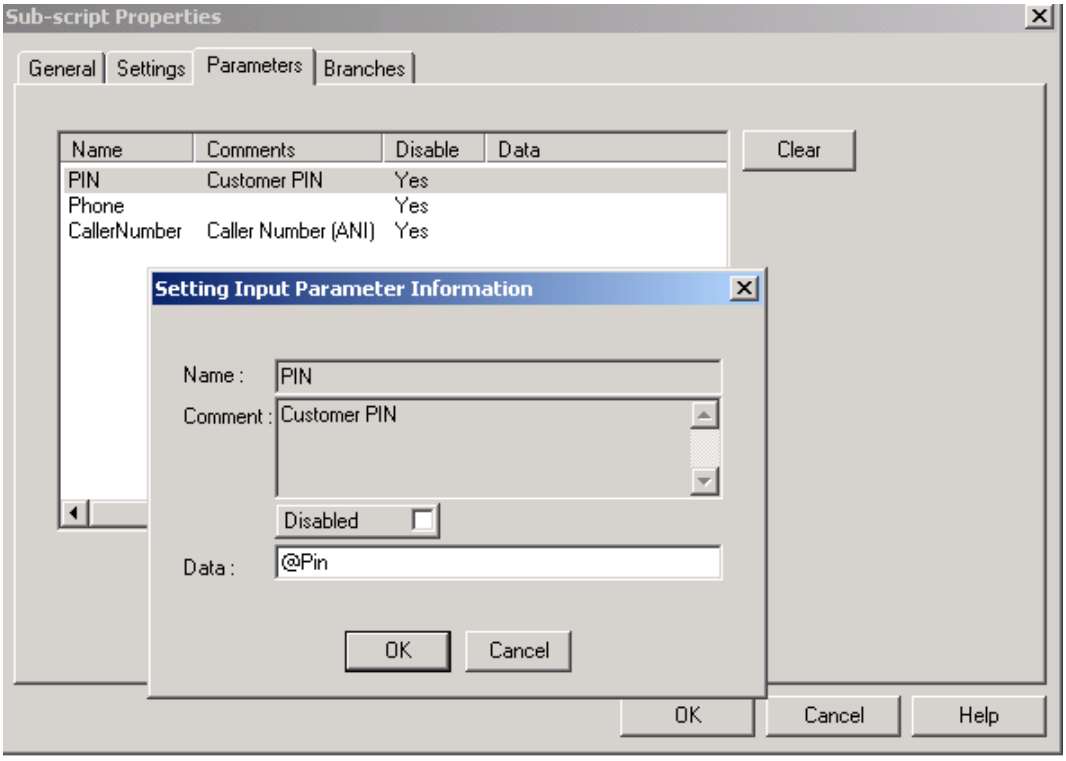


Figure 10: Setting Input Parameter Information

VARIABLE MAPPING

The second method for providing input data to the subscript is to use variable mapping. This enables the calling script to map a calling script variable to a subscript variable. The two variables do not need to have the same name, but must be of the same data type. The calling script and the subscript are referencing the same data. Changing the data in the subscript will be reflected in the variable when returned to the calling script.

Enable the **Configurable** option for the variable in the subscript.

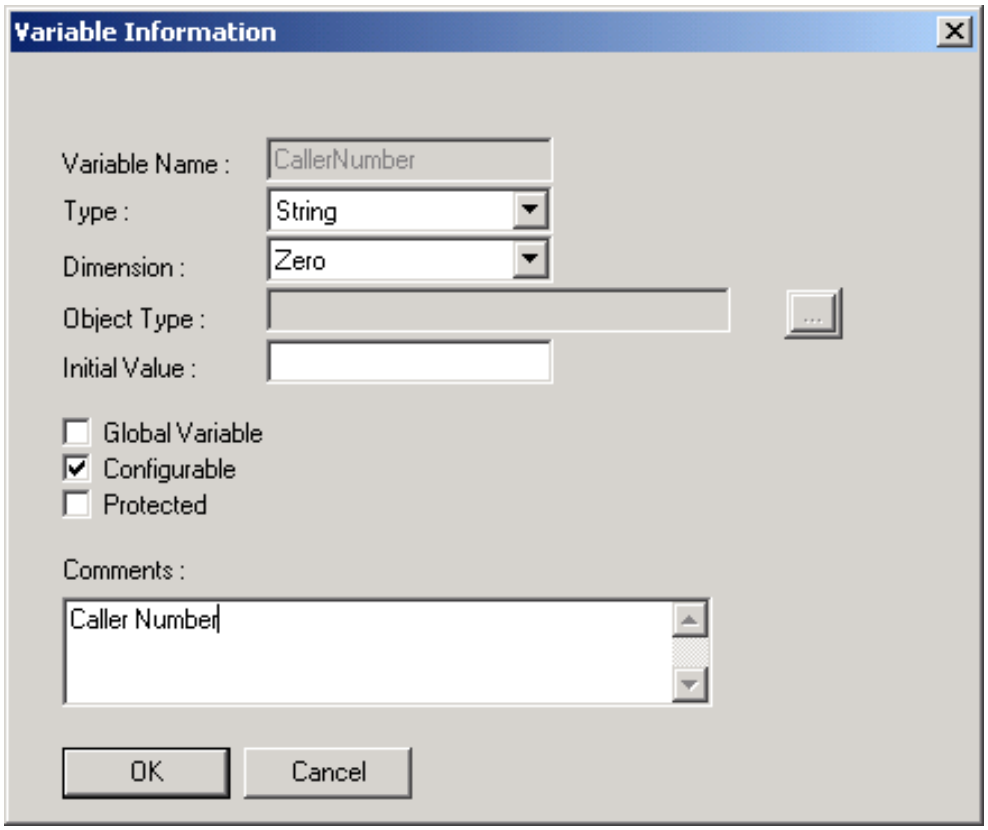


Figure 11: Variable Information dialog box

The configurable variables defined in the subscript will be displayed in the **Settings** tab of the **Sub-script Properties** block of the calling script. By clicking on Script Variable or Session Variable, it is possible to map a variable of the calling script to a configurable variable of the subscript. This enables the exchange of data between the calling script and subscript.

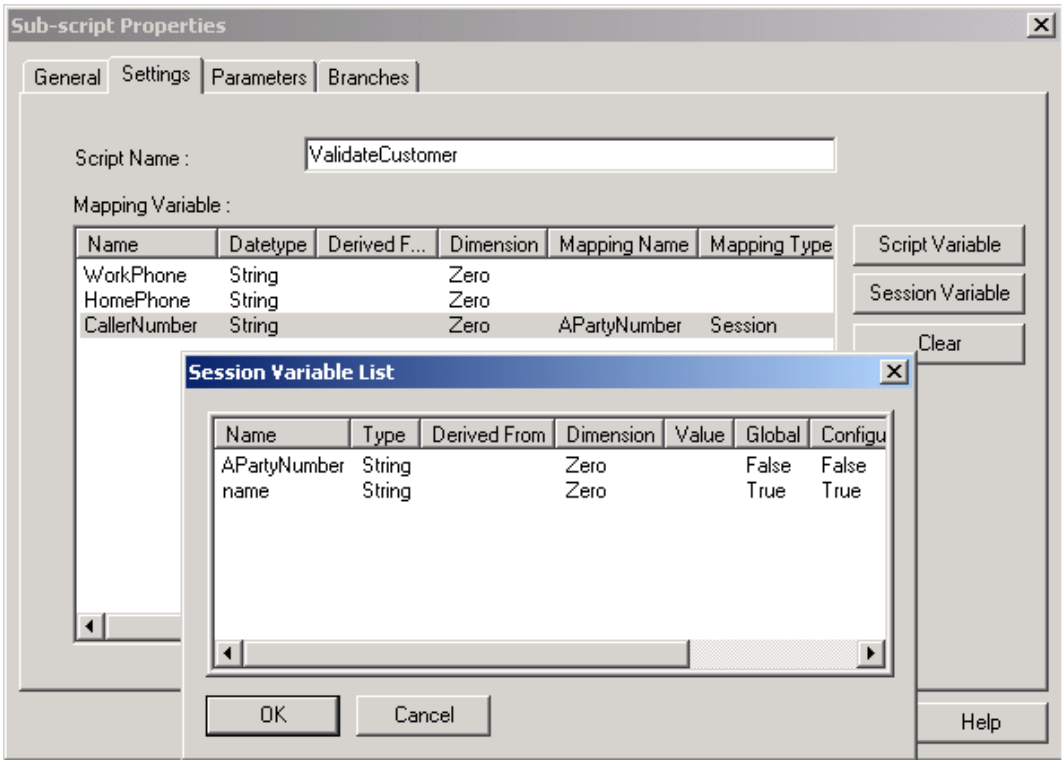


Figure 12: Session Variable list selected from Settings tab of Sub-script Properties

GLOBAL VARIABLES

The third method is to use global variables. Variables with the **Global Variable** option checked, the same name, and same data type defined in the calling script and subscripts are referencing the same data. Changing the data in the subscript will be reflected in the variable when returned to the calling script.

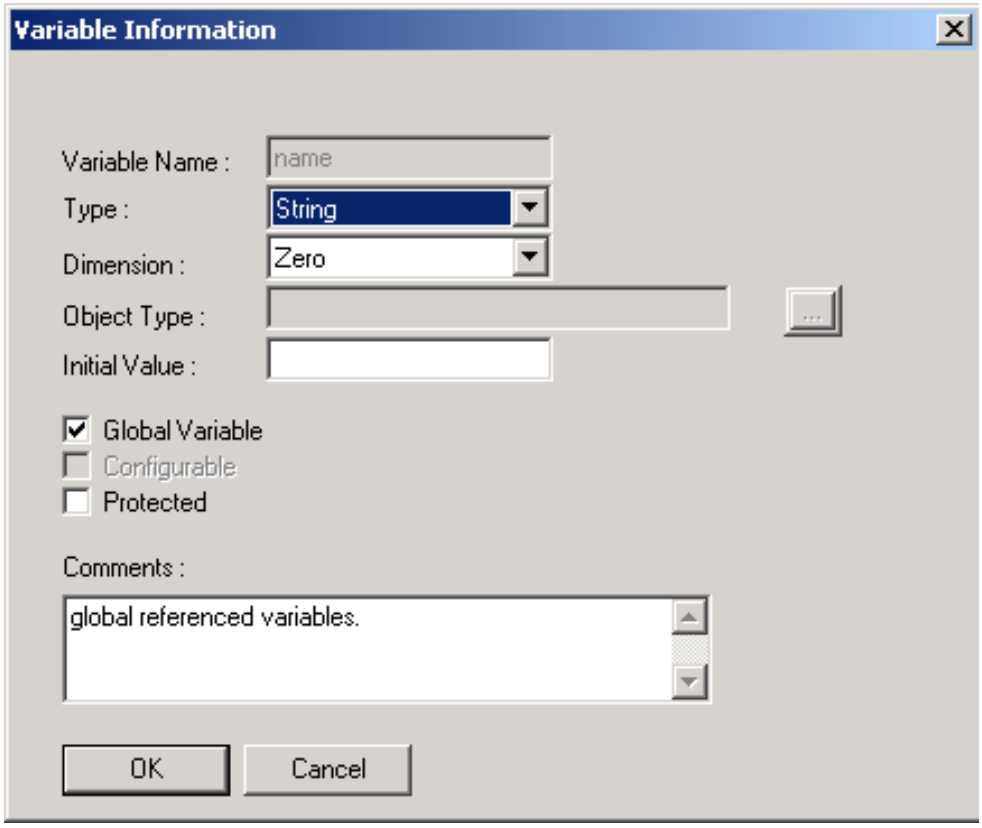
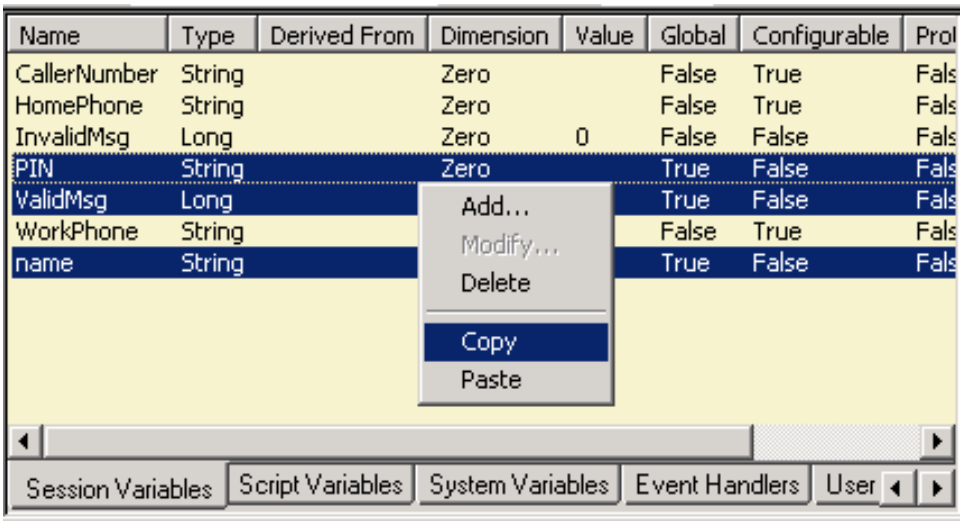


Figure 13: Global Variable option selected

Both the calling script and subscript must have **Global Variable** enabled. An easy way to configure it is to use the copy and paste feature.

1. Create the variables and select **Global Variable** on the calling script.
2. **Copy** one or more variables from the Object pane, Session Variable or Script Variable tab.



3. **Paste** on the Object pane, Session Variable or Script Variable tab to the script that you want the variables to be created for.

USING VARIABLE MAPPING IN A SUBSCRIPT

This tutorial handles using of variable mapping. The scripts used are `CallingValidateCustomer.mfd` and `ValidateCustomer.sfd`. The ValidateCustomer subscript validates the customer information via entered DTMF digits, and returns the name and the age of the caller. The call is then sent to a different service group based on the age group of the caller.

VALIDATECUSTOMER.MFD

This script returns four different results depending on the result of the database lookup. There are four return labels that indicate different results:

- Failure
- Child
- Senior
- Adult

The results will show in the subscript block on the calling script as the branch options.

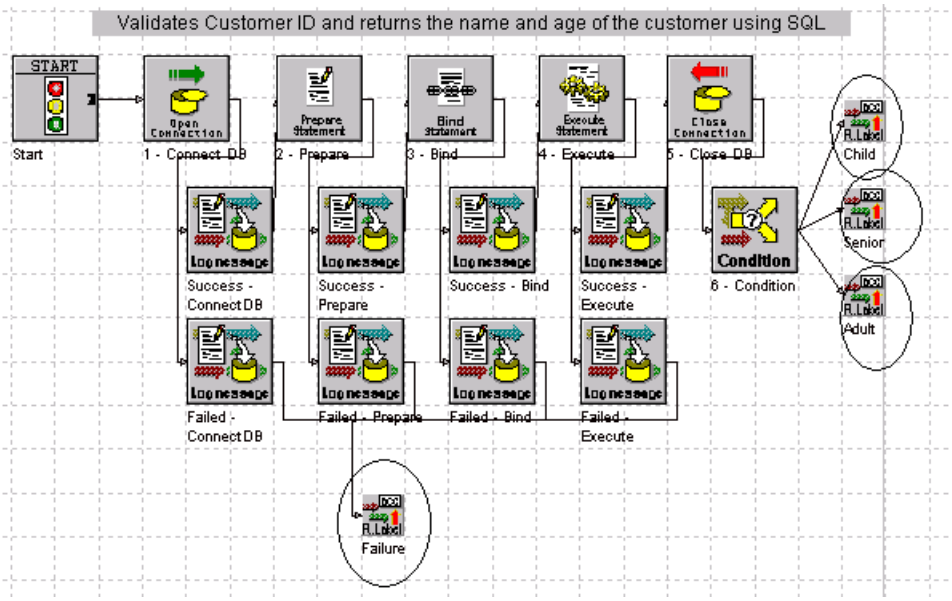


Figure 14: ValidateCustomer results

Input Parameter Settings

The script defines a Pin Input Parameter mapped to the Pin session variable in the subscript.

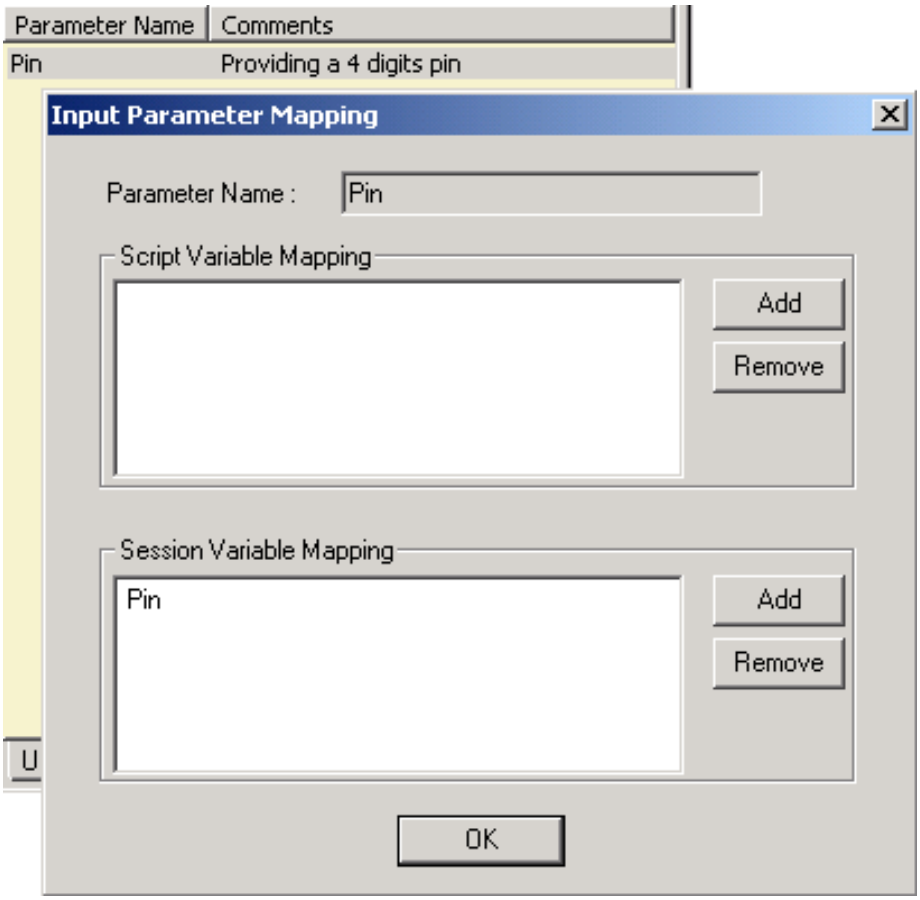


Figure 15: Pin Input Parameter

CALLING VALIDATECUSTOMER.SFD

The **Sub-script Properties** dialog contains four tabs:

- General
- Parameters
- Settings
- Branches

PARAMETER

The input to the subscript is set to the data in the variable @Pin.

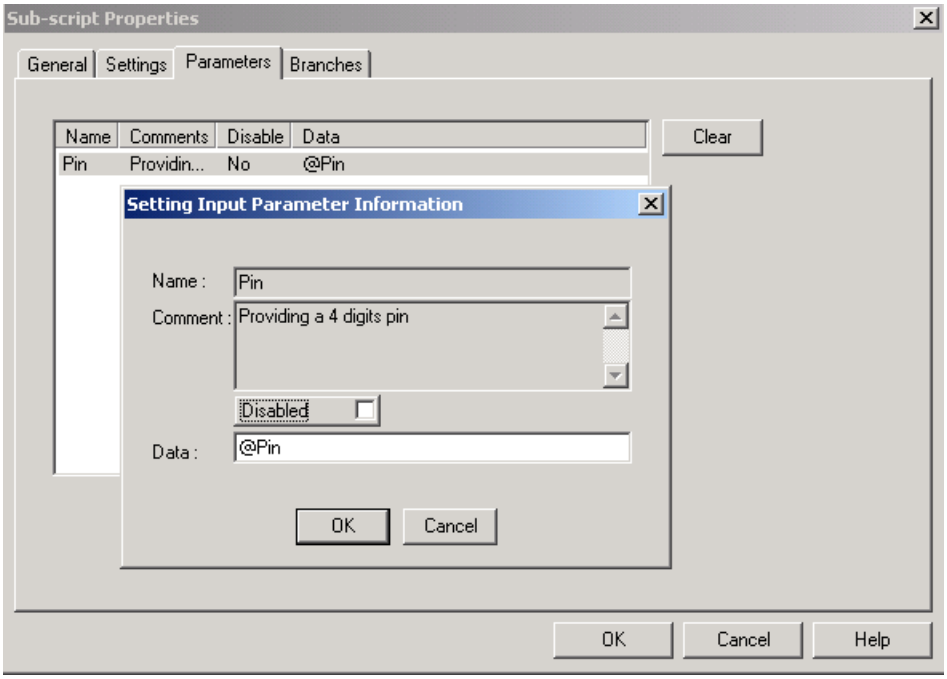


Figure 16: Select to set data in the variable @Pin

SETTINGS

The settings tab is shown in the figure below.

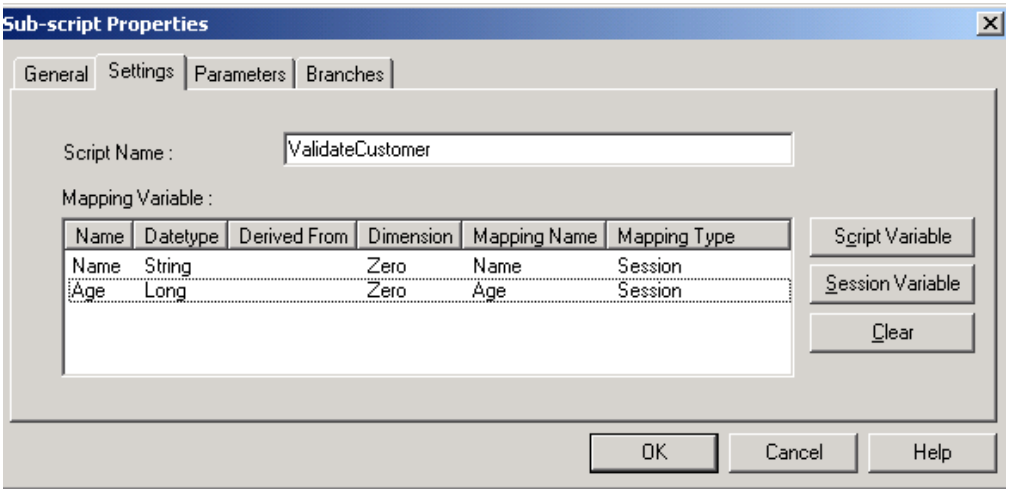


Figure 17: Settings tab of Sub-script Properties

BRANCHES

Name is mapped to the session variable Name, and **Age** is mapped to the session variable Age in the calling script. When a result is provided from the subscript, the value of Name and Age in the calling script will be updated. The branches provided in the subscript properties list the same number of subscript return labels used in the subscript.

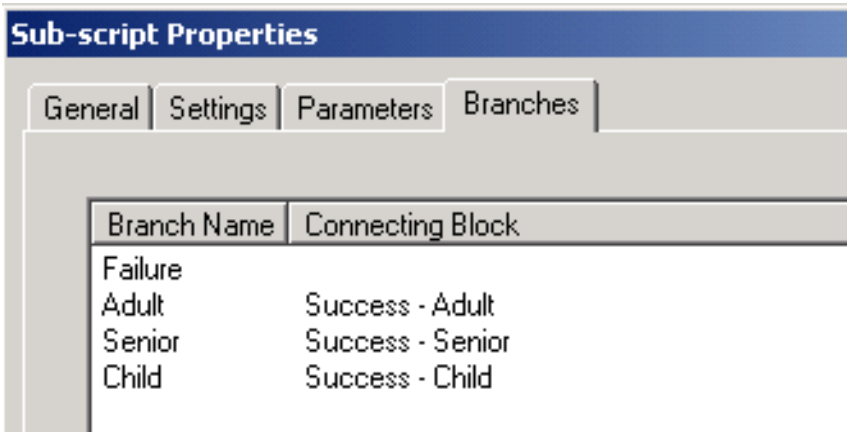


Figure 18: Branches tab of Sub-script Properties

USING GLOBAL VARIABLES IN A SUBSCRIPT

This tutorial handles the use of global variables in a subscript. The scripts used in the tutorial are `CallingValidateSTPCustomer.mfd` and `ValidateSTPCustomer.sfd`.

The ValidateSTPCustomer subscript validates the customer's information via caller entered DTMF digits, and returns name and the age of the caller. The call is then sent to a different service group based on the age group of the caller.

VALIDATESTPCUSTOMER.SFD

The ValidateSTPCustomer subscript has the same function as the ValidateCustomer subscript but it uses global variables. ValidateSTPCustomer provides the result of the name and age in the global variables.

Both the calling script and subscript have the global variables. Name and age will be updated when the subscript is completed with a successful result.

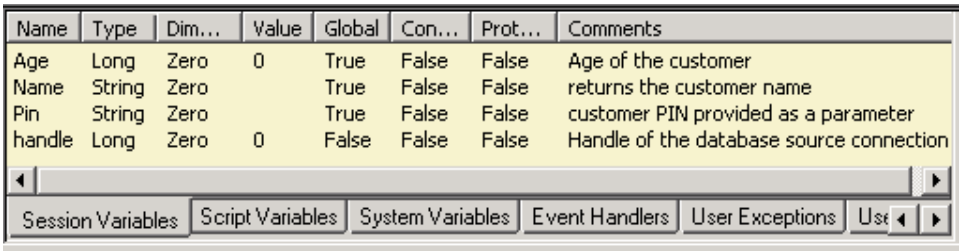


Figure 19: Session Variables

CALLINGVALIDATESTPCUSTOMER.MFD

In the **CallingValidateSTPCustomer** main script, **ValidateSTPCustomer** subscript is used to obtain customer specific information. If customer specific information cannot be retrieved, it will go to a service group to be handled by an agent.

The calling script is similar to the previous subscript example, except that the calling script must map the variables to a configurable variable in the subscript. A variable with the same name and with the **Global Variable** option checked needs to be defined. This enables the exchange of the data between the calling script and the subscript.

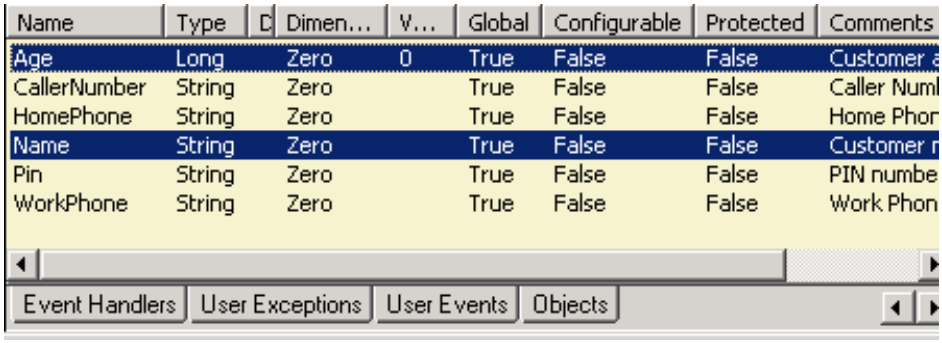


Figure 20

TESTING THE APPLICATION

The sample scripts use the Text-to-Speech feature to play the messages.

1. Make sure Text-to-Speech is installed and configured in the OAS server before using these sample scripts.
2. Create several service groups in Configuration Manager and modify the script to use the created service groups.
3. Recompile the script and create a Service Access using the compiled script.
4. Configure the monitor device via Configuration Manager.
5. Activate the service access and test run it.

TRIGGER AT STARTUP

It is not required for sessions to be started with an incoming call. A script that starts a session in the Event-Driven section when the service access is activated can be created. This tutorial provides an example of how to use this feature.

STARTUP SECTION

The Startup section is always executed when the Service Access is activated. From the Startup section, the script can trigger a new session in the event-driven section. Double clicking on the **StartupSection.mfd** opens the script. Select Startup section to display the script.

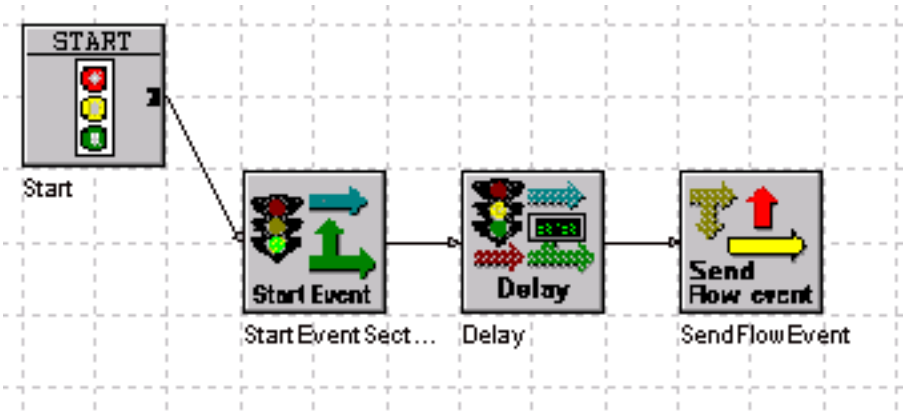


Figure 21: Startup script

The StartEventSection block indicates to start the Event-Driven section once it reaches the block. This will trigger the Event-Driven section to be started, and wait for a triggering event.

Delay

The Delay block delays the process for 5 seconds. This will give sufficient time for the Event-Driven section to be initialized and ready to receive events.

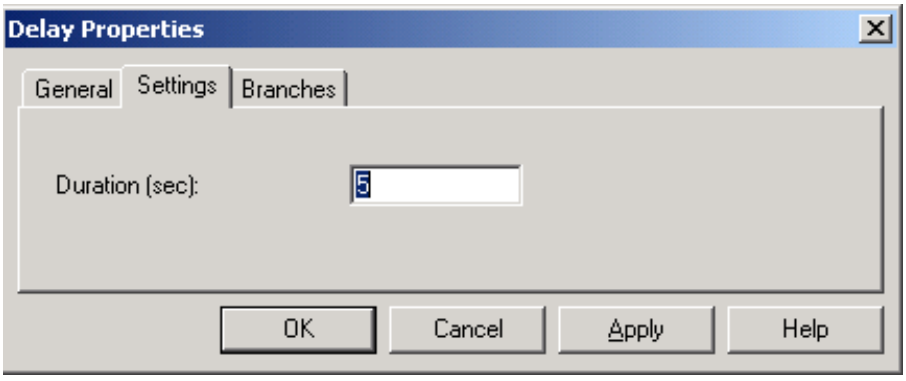


Figure 22: Settings tab of Delay Properties

SendFlowEvent

The SendFlowEvent block sends an event to the Event-Driven section with the Event Name called “OutgoingCall” and some data. The Event Data is provided to the Event-Driven section as the input data.

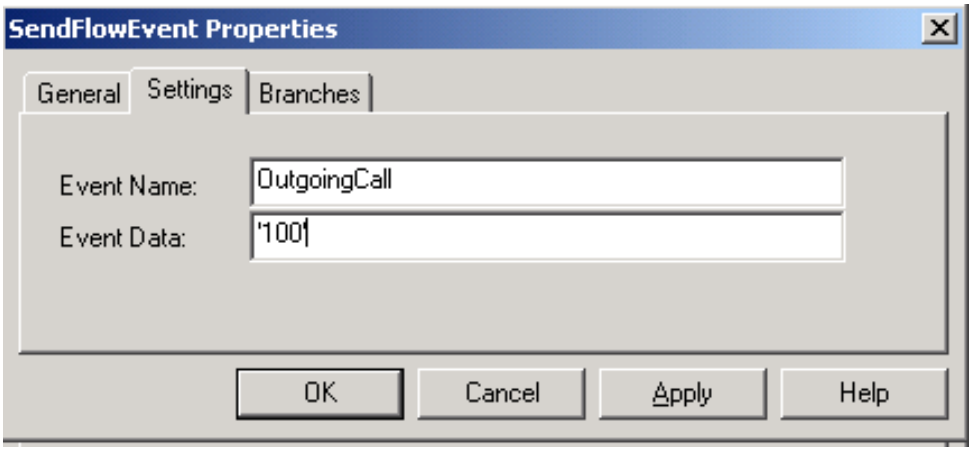


Figure 23: Event name and data, SendFlowEvent

EVENT DRIVEN SECTION

Select the Event-Driven section to switch to the Event-Driven script.

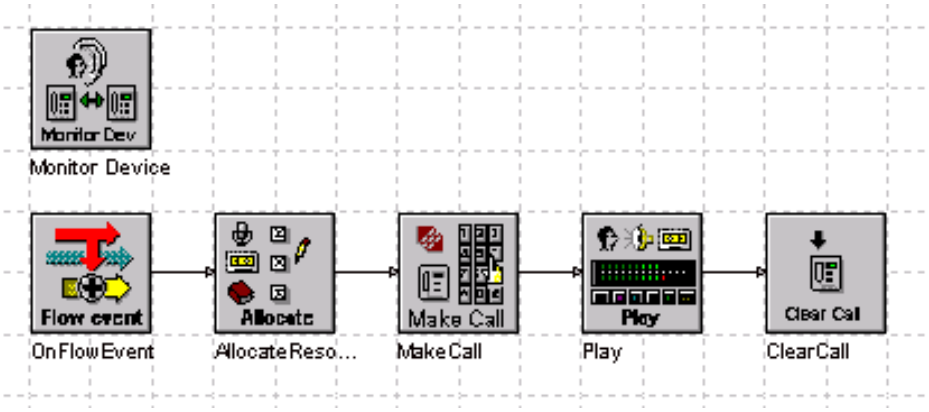


Figure 24: Event-Driven script

OnFlowEvent

The OnFlowEvent block waits for an event called “OutgoingCall” and saves the data to the variable @Data. Since the event was sent from the startup section, this triggers the session to start and immediately process the allocated resource block. In this example, the script will make an outgoing call to a number, play a message and then hang up. This example uses the @Data as the called number in the **MakeCall** block.

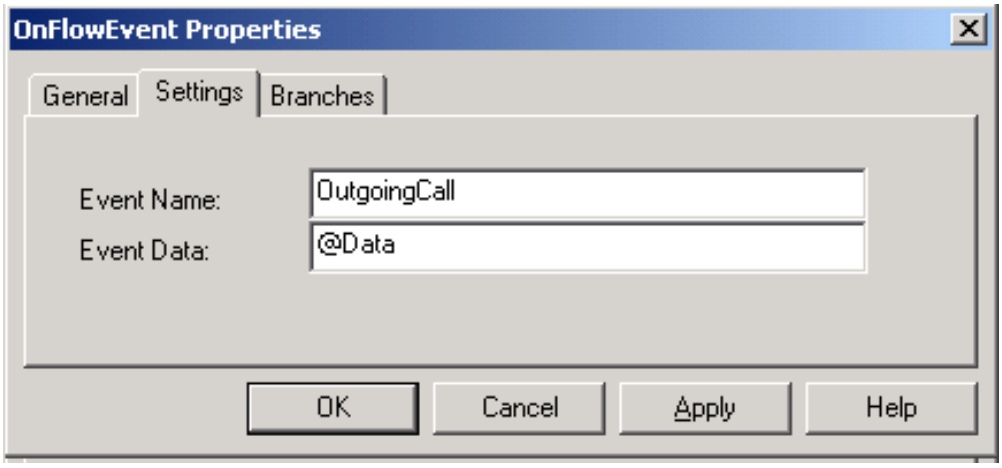


Figure 25: Event name and data, OnFlowEvent

MonitorDevice

A MonitorDevice is needed for this scenario to make the outgoing call. The MonitorDevice block is added to the script and does not need to be connected to a block.

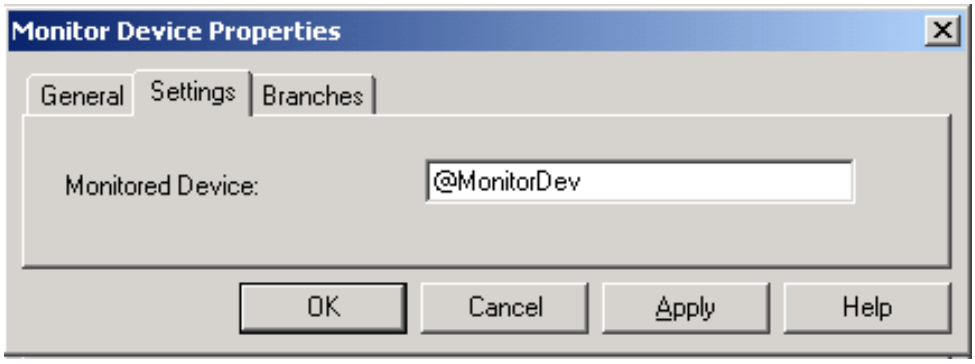


Figure 26: Set Monitored Device

AllocateResources

In this example, the AllocateResource block **Allocate Resources for Future Call** option is selected. Since there is no existing call when this script is running, it must be set for a future call. The Device Name must be entered to make a future call and should be the same device as in the MonitorDevice block.

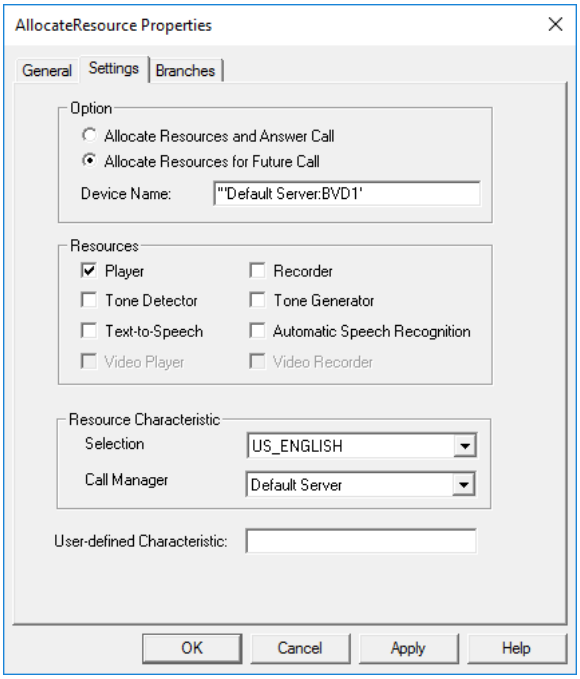


Figure 27: Allocate Resource for Future Call selected

Device Name in the Allocate Resource for Future Call

The device name is used when allocating a resource used in a future outbound call. The device name must be enclosed in single or double quotes, or be a variable containing the BVD name.

MakeCall

The **MakeCall** block makes an outgoing call from the BVD specified in the AllocateResource block. It makes a call to the device specified in the @Data variable and waits for the called party to answer. If the called party does not answer within the Answer Time-out, the block will clear the call and branch to the No-Answer branch. In the case that the called party answers, it will play the specified message and clear the call afterward.

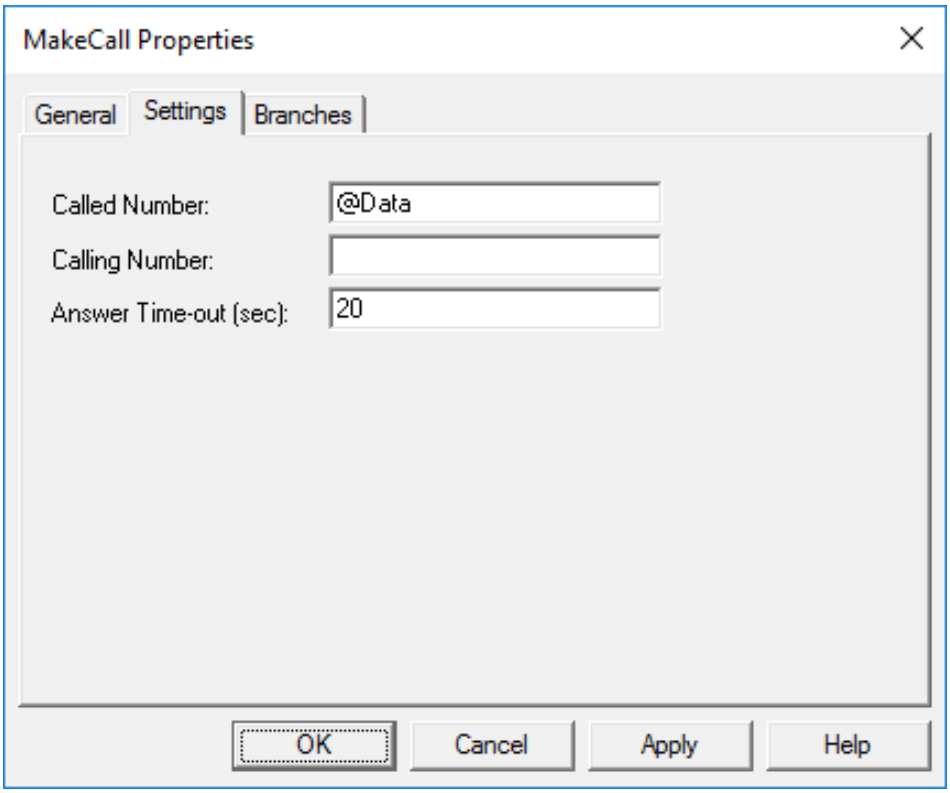


Figure 28: Settings tab of MakeCall Properties

VOICE PROMPTS

One voice prompt is used in this script to play a message to the called party. Make sure the voice prompt is available and is configured in OAS as a play message.

Table 2: Voice Prompts to called party

SCRIPT MANAGER VARIABLE	MESSAGE ID	SAMPLE VOICE PROMPT DESCRIPTION
Msg	1210	“Welcome to… Goodbye.”

TESTING THE APPLICATION

Create a Service Access using the compiled script. It is very important to use Configuration Manager or Script Manager Configuration to configure the called device, the monitor device and the play message or the Service Application will not be able to start or run properly.

Activate the Service Application and test it.

SYSTEM EVENT HANDLER

In this tutorial, the `OverrideEventHandler.mfd` script is described.

OVERRIDEEVENTHANDLER.MFD

This script uses the Block Not Connected event handler to demonstrate the definition of Event Handler and the use of the OverrideEventHandler block. The objective of this script is to log different messages depending on when the Block Not Connected event handler is thrown in a call flow.

First, the Block Not Connected event handler method is defined to execute the 'Pre Main Menu' logmsg block.

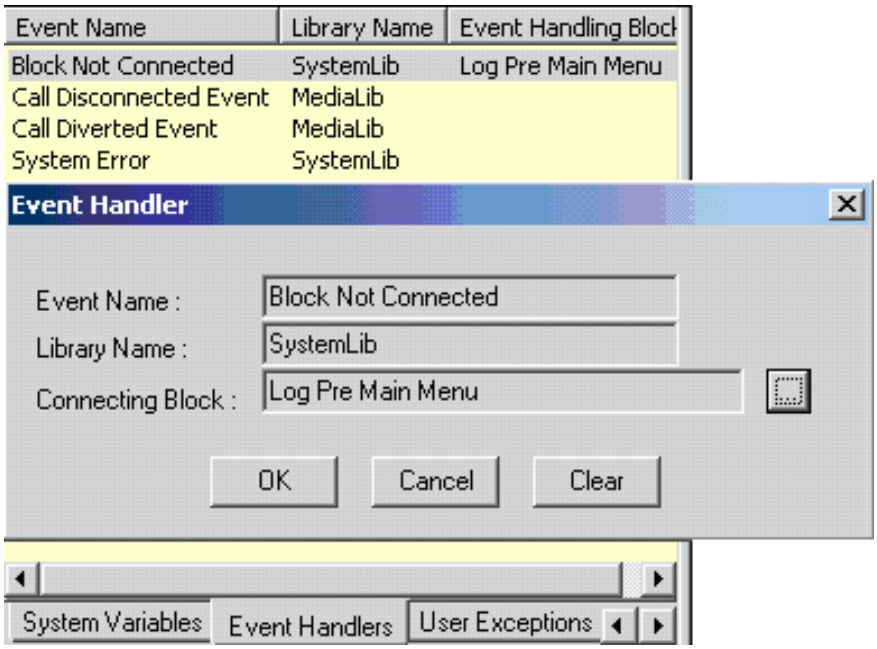


Figure 29: Event Handler defined to execute Pre Main Menu

In the event driven section of the script, the OverrideEventHandler block redefines the handling method for the Block Not Connected Event. It is set after the first main menu.

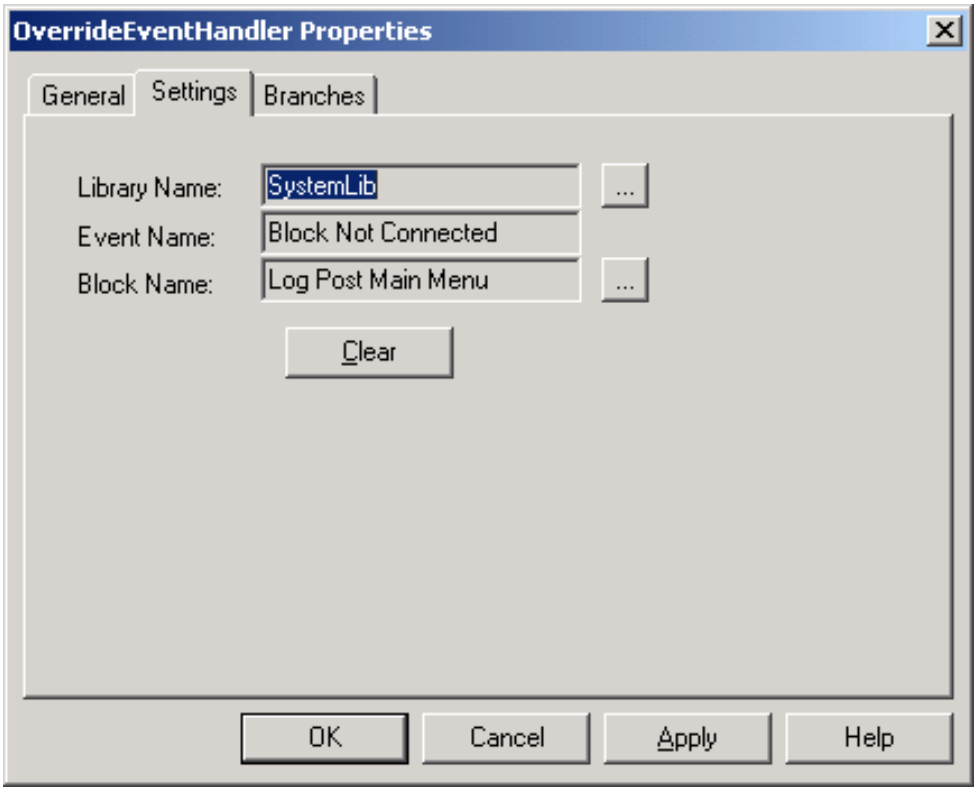


Figure 30: Settings tab of OverrideEventHandler Properties

If the caller selects an undefined branch from the first main menu block, the Log Pre Main Menu block will be executed. If the caller selects an undefined branch on the second menu selection, the Log Post Main Menu block is executed instead.

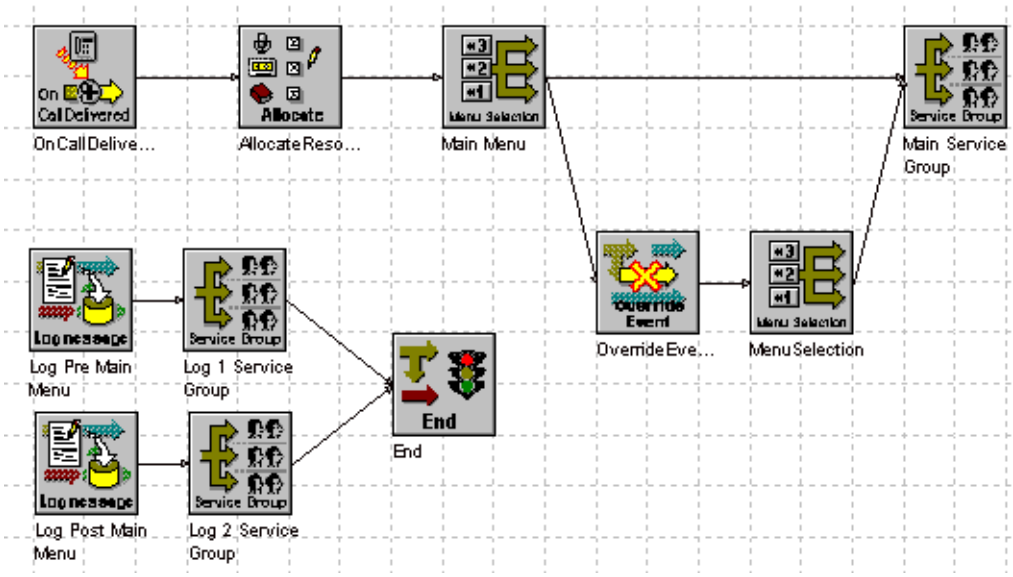


Figure 31: Log Pre Main Menu and Log Post Main Menu

VOICE PROMPTS

The voice prompts and play messages must be configured in the OAS server before running the script.

Table 3: Voice Prompts System Event Handler

SCRIPT MANAGER VARIABLE	MESSAGE ID	SAMPLE VOICE PROMPT DESCRIPTION
Msg1	1211	“Main Menu… Please press 1.”
Msg2	1212	“Second Menu… Please press 1.”

TESTING THE APPLICATION

To test the application:

1. Create a service access via Configuration Manager to use the compiled user exception script
2. Activate the service access
3. Make a call into the script and press 5 in the main menu
4. Check the event viewer for information
5. Repeat the procedure, but this time press 5 in the second menu selection and check the event viewer again.

USER EXCEPTION

This tutorial demonstrates the use of SetUserExceptionHandler, ThrowUserException and UserExceptionHandler. The UserException component works much like the Goto block but with the advantage that it works across the whole project including subscripts.

Common usages of these components are:

- Allow callers to exit a script quickly to a service group
- Allow callers to access a main menu from any subscript

SUBUSEREX.SFD

A subscript called subuserex.sfd is already created. The User Exception subscript contains a MenuSelection block which allows the user to select to throw a user defined Exception block or just return to the main script.

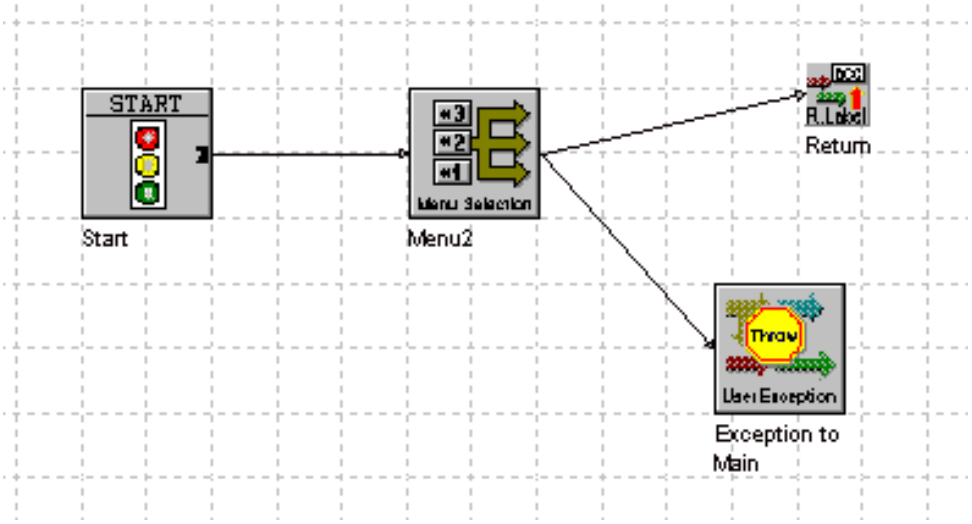


Figure 32: SubUserEx script

USER EXCEPTION

The ThrowUserException block throws a user defined exception named **Ex_Main** whenever the block is executed. The flow processor first searches the user defined exception handler within the subscript. Since the Ex_Main user exception is not defined in this subscript, the exception will be passed up to the calling script in search of the Ex_Main user defined exception.

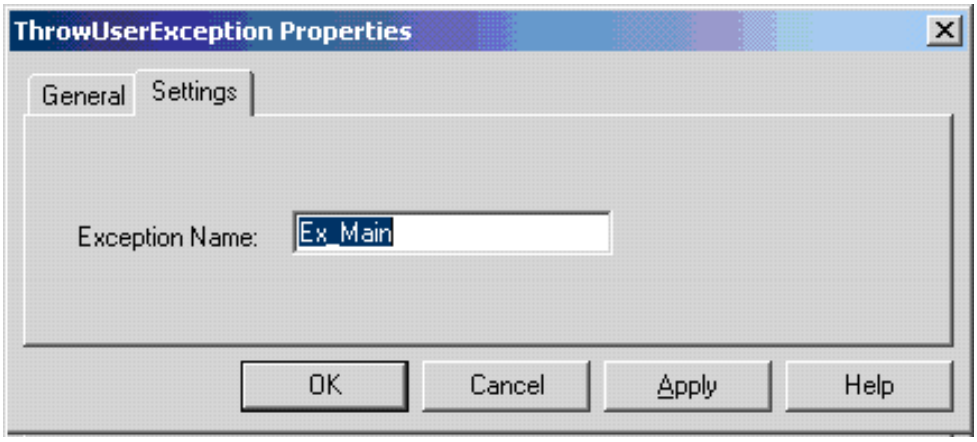


Figure 33: ThrowUserException

USEREXCEPTIONMAIN.MFD

The UserExceptionMain main script contains a selection menu. The **MenuSelection** block allows the caller to branch to another **ThrowUserException** block (option 0), to branch to the subscript created above (option 1), and to a Service Group (option 2).

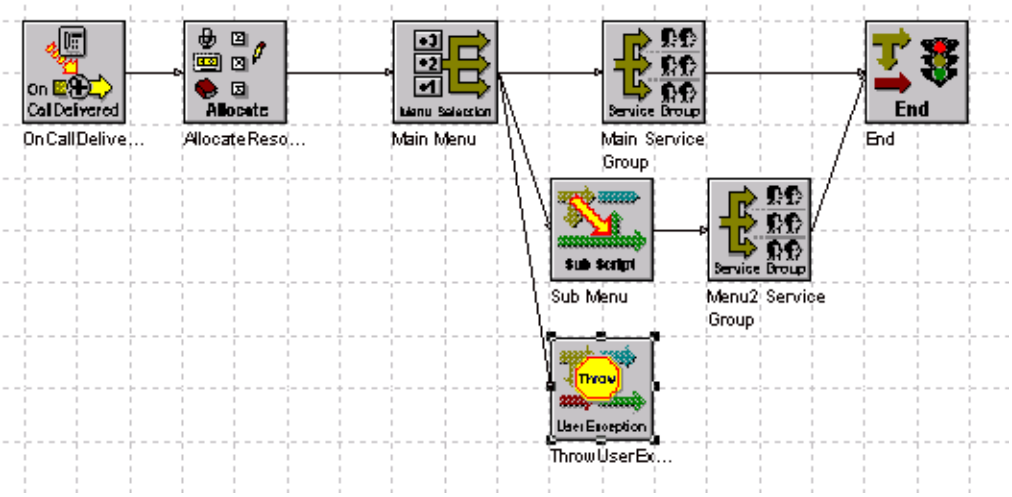


Figure 34: UserExceptionMain script

The UserException defines the exception name Ex_Main. It defines the block to go to when the exception is thrown, Main Service Group in this example.

Figure 35: User Exception Name

VOICE PROMPTS

The voice prompt and play messages must be configured in the OAS server before running the script.

Table 4: Voice Prompts

SCRIPT MANAGER VARIABLE	MESSAGE ID	SAMPLE VOICE PROMPT DESCRIPTION
Msg3	1213	"To throw an user exception, press 0, to go to the sub-script, press 1, to go to a service group, press 2".

TESTING THE APPLICATION

To test the application:

1. Create a service access in Configuration Manager to use the compiled script
2. Activate the service access
3. Make a call into the script and press 0 at any time for the call to be routed to the Main service group

The User Exception is always consumed whenever a valid **ThrowUserException** block is used. For the second example, callers are allowed to return to a main menu selection from any part of the script. A different method of defining User Exception is required in order to reset a consumed User Exception.

Note: A system error event handler will be thrown if an invalid ThrowUserException (an undefined user exception) block is executed.

VOICE PROMPTS

The voice prompts and play messages must be configured in the OAS server before running the script.

Table 5: Voice Prompts

SCRIPT MANAGER VARIABLE	MESSAGE ID	SAMPLE VOICE PROMPT DESCRIPTION
Msg3	1213	"Welcome… Please press 1 for … and 0 at any time to speak to an agent."
Msg4	1214	"Please press 1 for …"
Msg5	1215	"Welcome…Please press 1 for … and 0 to return to the main menu."

USEREXCEPTIONREPEAT.MFD

Open the UserExceptionRepeat.mfd script for this tutorial. This script is similar to the UserException.mfd script with one exception - the User Exception is defined using SetUserException block.

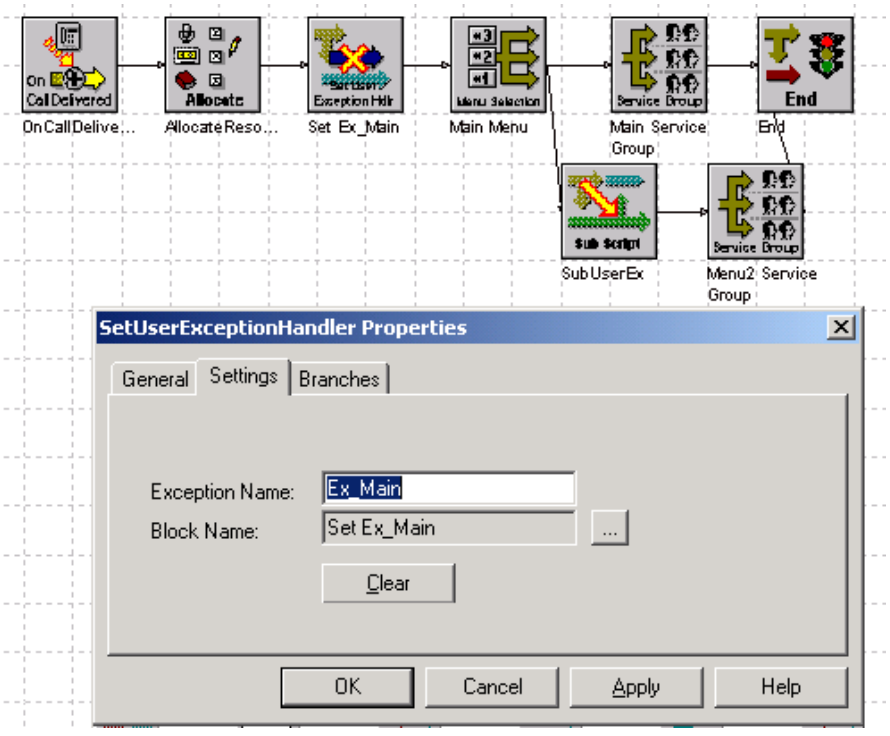


Figure 36: User ExceptionRepeat script

When **SetUserException** block is first executed, the "Ex_Main" user exception is defined for the session. Once a **ThrowUserException** block is executed within the same session, the call will go back to the **SetUserException** block and reset the "Ex_Main" user exception so that it can be reused within the same session.

TEST THE APPLICATION

The same subscript as the previous tutorial, `SubUserEx.sfd`, is also applicable in this tutorial since the name of the user exception is the same.

Using the same service access created in the earlier user exception tutorial, change the service access to use **UserExceptionRepeat** and restart the service access. Make a call into the application and press 0 at any time during the script and the call will return to the main menu.

Note: The caller can repeatedly press 0 and the script will return to the same main menu.

USER EVENT AND GLOBAL KEY

A subscript or subroutine within a script can be triggered using the **SendUserEvent** block and returns to its originating or calling point when **Event Continue** block is used.

The **SendUserEvent** block is especially useful when there is a set of common global keys defined for a project and a Help routine is required to play global key usage from any part of the call flow.

In this tutorial, the **Help** subscript will be used to play a message that explains the definition and usage of each global key and then returns to the originating calling point once the subscript is completed.

HELP.SFD

A subscript called `help.sfd` has already been created. This subscript contains a **Play** block that plays a help message and returns to the calling script.

Figure 37: Help.sfd

USEREVENT.MFD

The `UserEvent.mfd` main script is a simple script with a **Branch** block that branches to a **SendUserEvent** block when a global key is detected. It is then followed by a **Service Group** block.

Store Global Keys

A variable called @GlobalKey is created to store global keys. Global keys can be one or more sequences of digits.

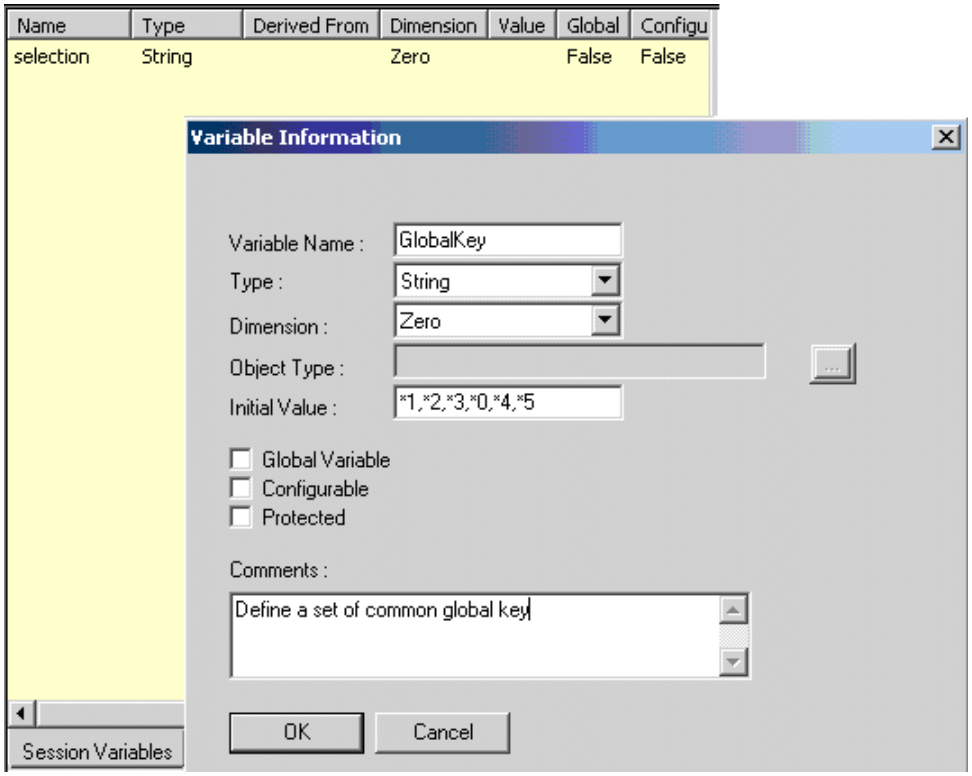


Figure 38: Global Key

In the same main script, a subscript block is used to call the **Help** subscript. In order to return the flow back to its originating point after the completion of Help subscript, an **Event Continue** block is added at the end of the subscript Help block.

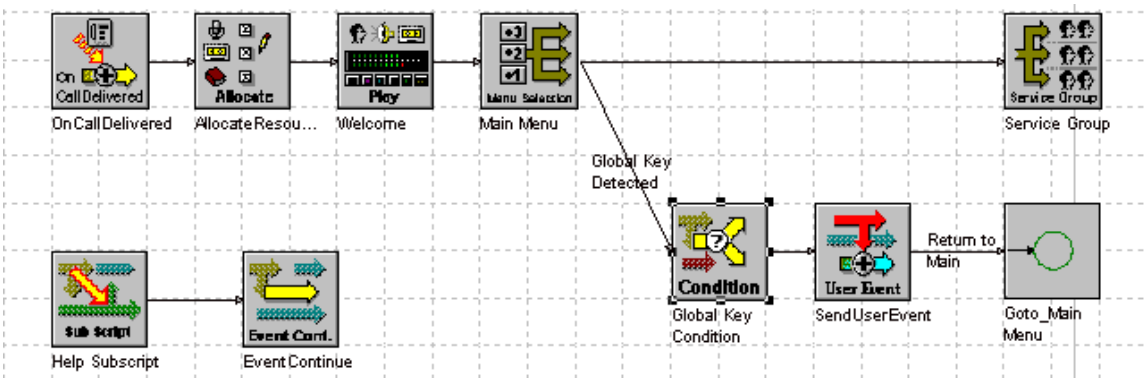


Figure 39: UserEvent.mfd

Note: **EventContinue** only works in the same script where the handler starts or is defined.

Define user event

A user event named Evt_Help is defined to branch to the Help Sub-script block when the event is sent.

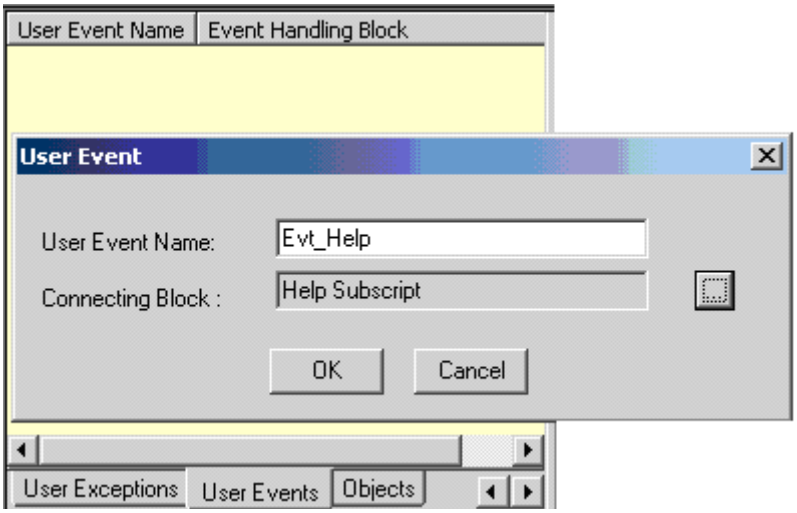


Figure 40: Define user event

In the **SendUserEvent** block, enter **Evt_Help** for the user event that was defined earlier into the User Event field.

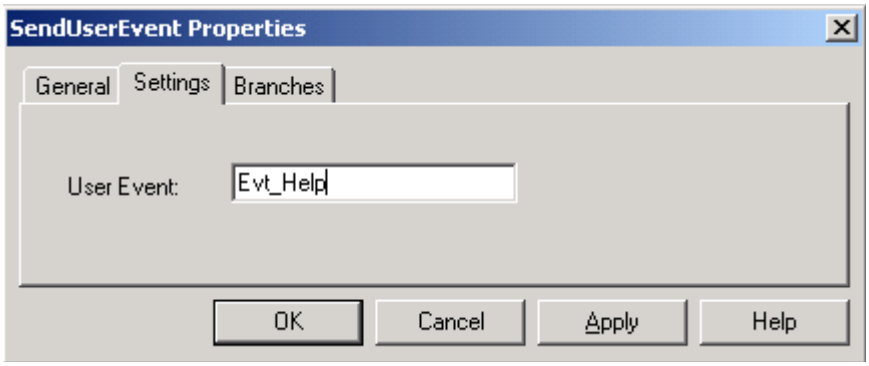


Figure 41: User event

When **SendUserEvent** block executes, it sends out the **Evt_Help** user event. The **Flow Processor** catches the user event and determines if it is a valid event defined in the script and triggers the execution of the **Help** block as the next block in the call flow.

USEREVENTWITHSUB.MFD

The user event defined in a main script can also be used in a subscript without redefinition of the user event handler in the subscript. When an undefined user event is used in a subscript's **SendUserEvent** block, the **Flow Processor** will automatically traverse up to the calling script in search of a matching user event.

The main script is modified to call a sub script named **SubTest**.

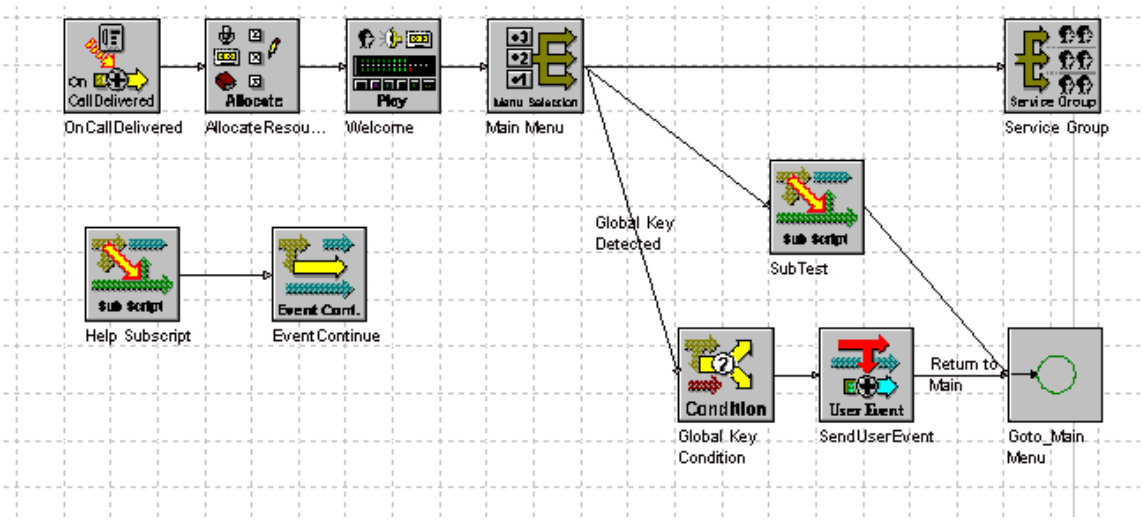


Figure 42: UserEventWithSub script

SUBTEST.SFD

In the **SubTest** subscript, the **SendUserEvent** Block is used in the subscript within the same project even though there is no defined User Event in the sub script.

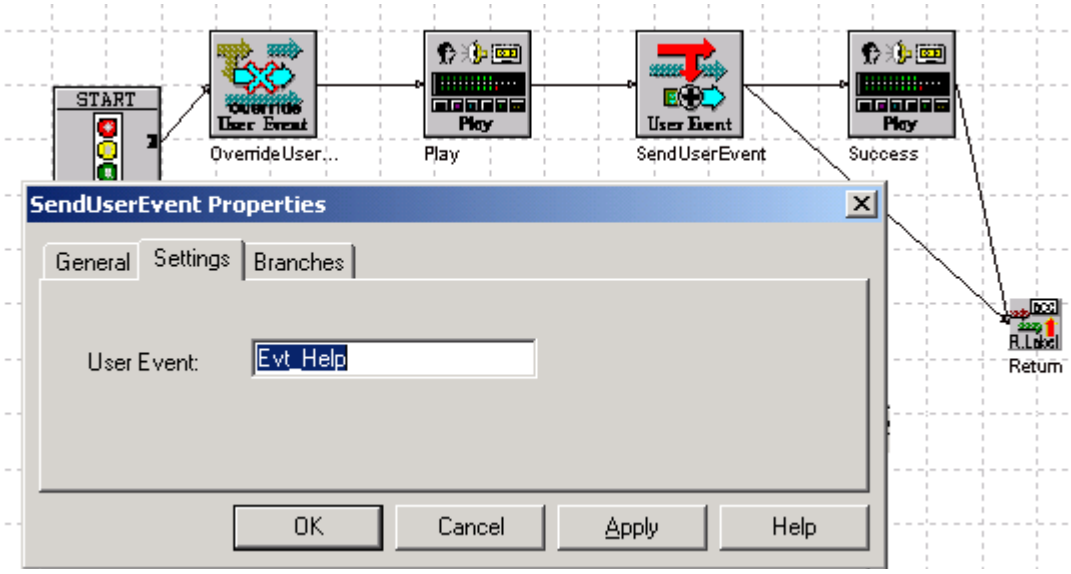


Figure 43: SendUserEventProperties

Test the call flow by selecting the digit to execute subscript SubTest from the main script. The same Help subscript will be executed when SendUserEvent block is executed in the SubTest subscript.

Override User Event

Override User Event is used if there is a different Help message to be played on a particular subscript. In this case, subscript SubTest is modified to use OverrideUserEvent block to overwrite the subroutine to be called when Send User Event is called.

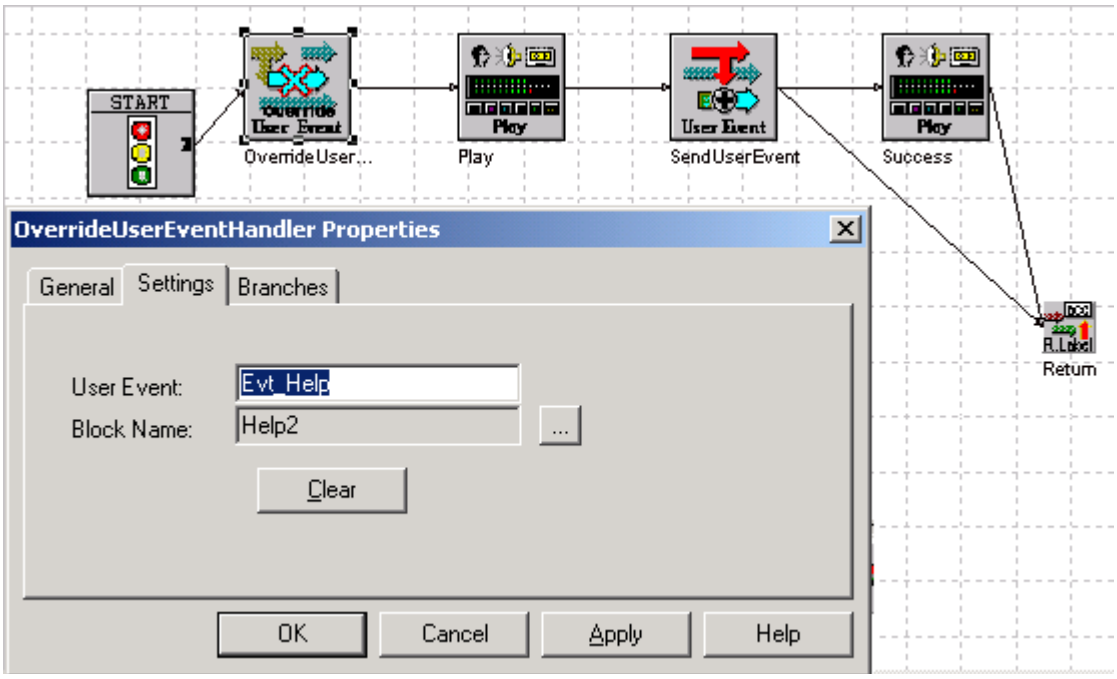


Figure 44: OverrideUserEvent

In this **SubTest** subscript you will find that the **Help2** play message is executed instead of the old Help subscript being called.

VOICE PROMPTS

The voice prompts and play messages must be configured in the OAS server before running the script.

Table 6: Voice prompts and play messages

SCRIPT MANAGER VARIABLE	MESSAGE ID	SAMPLE VOICE PROMPT DESCRIPTION
MsgHelp	1220	“Welcome to the automated system… For Service, press 1…”
MsgHelp2	1221	“This is an alternate help menu…”

JSCRIPTEXECUTE

The JScriptExecute block component in Script Manager has been developed using Mozilla's SpiderMonkey C implementation of JavaScript. For more information about the SpiderMonkey's JavaScript see the following link: *http://www.mozilla.org/js/spidermonkey/* .

The JScriptExecute component can use pure JavaScript language. Replication of the behavior of an ActiveXObject is not part of the script engine. This means that it is not possible to use ActiveXObjects to use Windows Jscript functionality. To make use of full compatibility under Windows for JScriptExecute, COM integration should be added.

This tutorial demonstrates the use of the JScriptExecute and the Object data type. The JScriptExecute block accesses the variable that was defined in the script. The block manipulates the data, formats it to a string and sends the text string to MiContact Center Agent.

JSCRIPT.MFD

From Script Designer, open the tutorials project and click on **Jscript** to open the script.

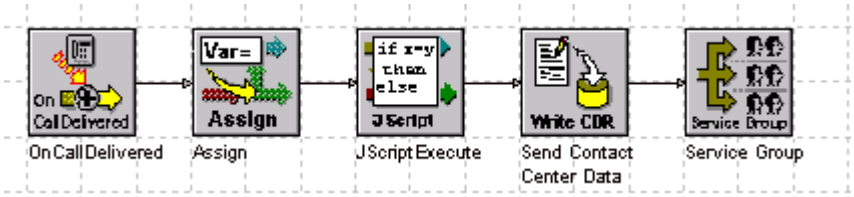


Figure 45: JScript.mfd

From the Script Object Pane, select the **Objects** tab, the **Object Information** dialog appears.

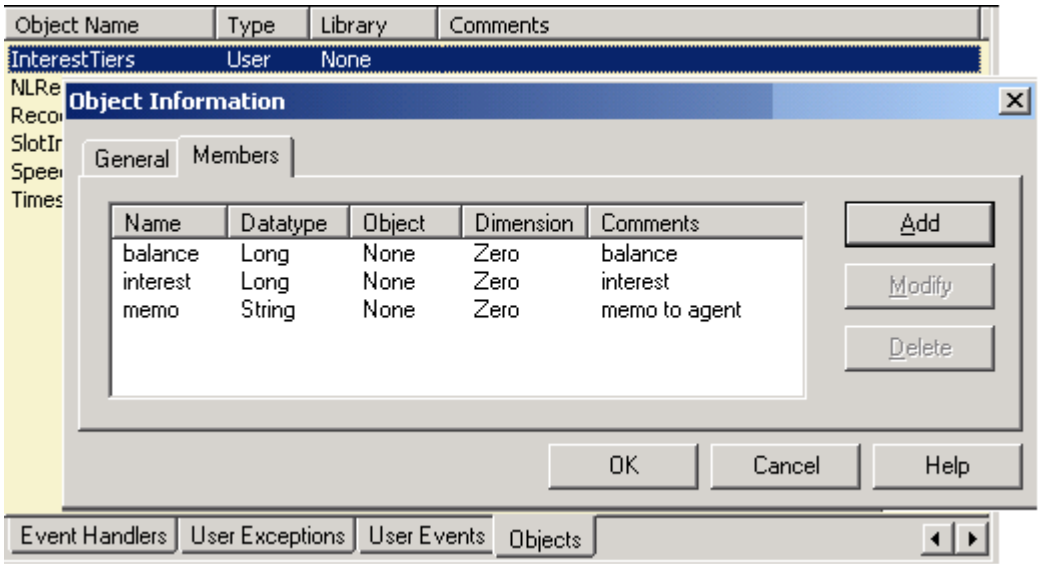


Figure 46: Object Information

A user defined object called InterestTiers is defined. Double click on InterestTiers and the Object Information dialog displays the members that are defined. In this example, there are three members defined: balance, interest and memo. Click on the Session Variables tab. A variable called Interest is defined as a one dimensional array of the InterestTiers data type.

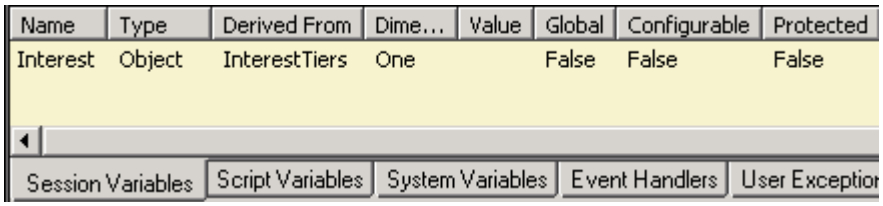


Figure 47: User defined object

After receiving an incoming call the script uses the Assign Block to set the balances.

ASSIGN BLOCK

The **Assign** block sets the balance of each element in the @Interest array to a different value.

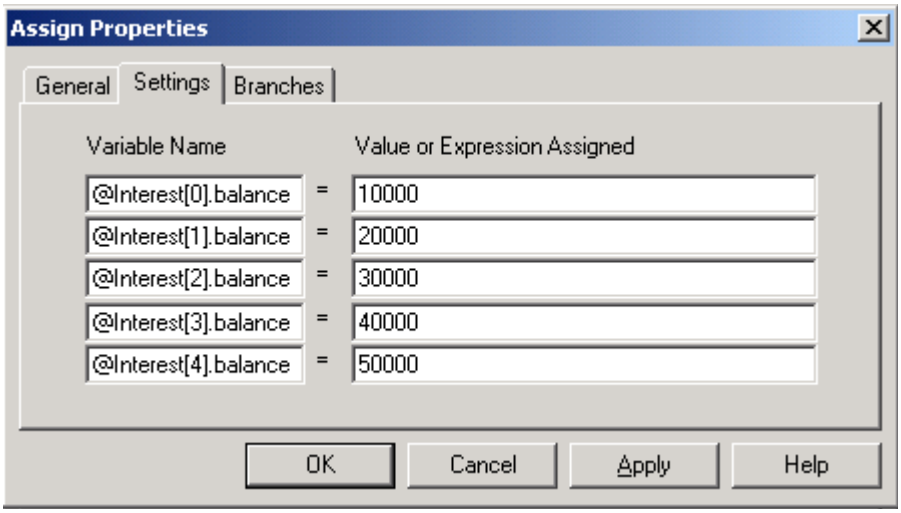


Figure 48: Assigning values to variables

Note: **EventContinue** only works in the same script where the handler starts or is defined.

JSCRIPTEXECUTE

The settings tab of the JScriptExecute Properties is shown in the figure below.

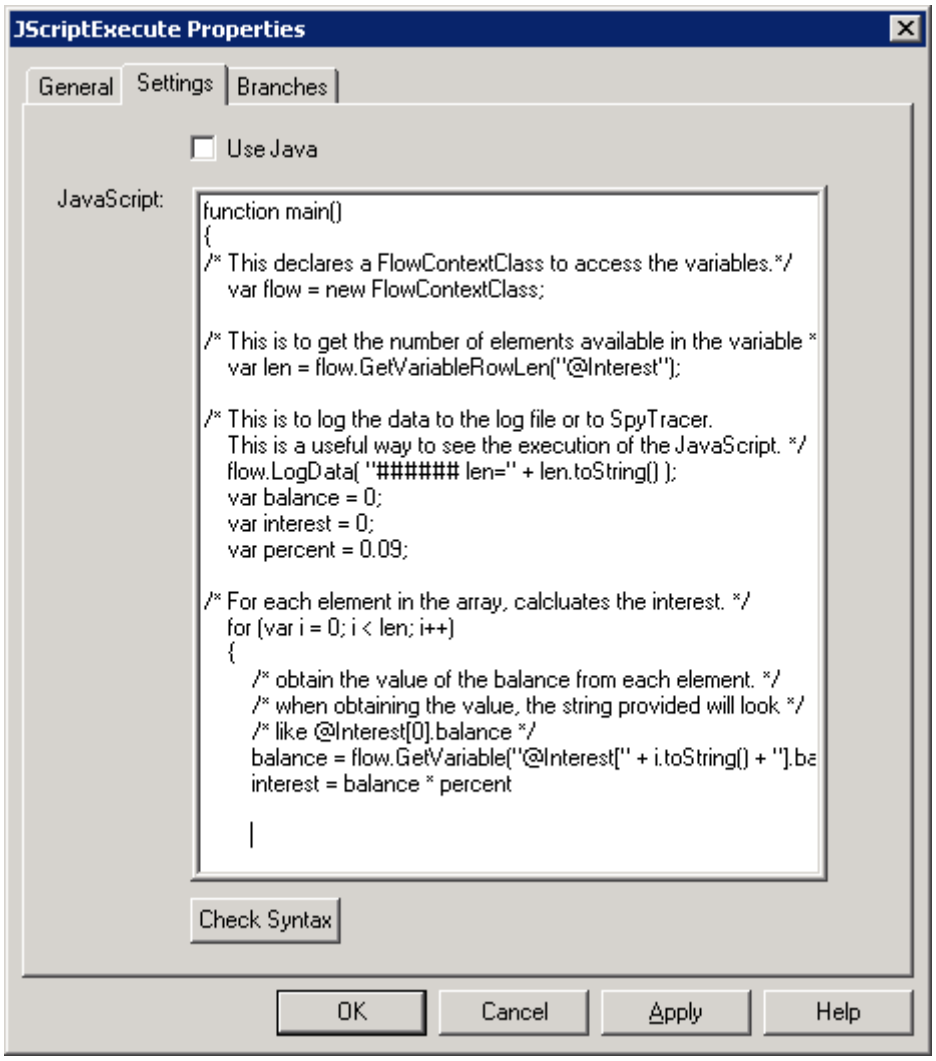


Figure 49: JScriptExecute settings

```
{/* This declares a FlowContextClass to access the variables. */

var flow = new FlowContextClass;

/* This is to get the number of elements available in the variable
@Interest.*/

 var len= flow.GetVariableRowLen("@Interest");
```

```
/*This is to log the data to the log file or to SpyTracer. This is a
useful way to see the execution of the JavaScript. */

flow.LogData( "###### len=" + len.toString());

var balance=0;

var interest=0;

var percent = 0.09;

/* For each element in the array, calculates the interest. */

for (var i=0; i<len; i++) {

      /* obtain the value of the balance from each element. */

      /* when obtaining the value, the string provided will look */

            /* like @Interest[0].balance */

            balance = flow.GetVariable("@Interest[" + i.toString()+
            "].balance");

            interest = balance * percent;

            percent = percent - 0.01;

            /*Logs the calculated interest */

            flow.LogData( "###### interest=" + interest.toString() );

            /* save the calculated interest to the
            @Interest[i].interest variable */

            flow.SetVariable("@Interest["+i.toString()+"].interest",
            interest);

            /* now prepare the data to present to the agent */

            var memo = balance.toString() + ","+ interest.toString();

            /* save the data to the @Interest[i].memo */
```

```
            flow.SetVariable("@Interest["+i.toString()+"].memo", memo);

}

/* set the result to 0 */

flow.SetResult(0);

}
```

USE JAVA OPTION

The checkbox **Use Java** in the Settings tab of JscriptExecute properties window enables method calls to Java. It enables the script to access standard Java class objects, that is, the Java classes that are part of the standard Java package.

Note: Access to user defined Java class objects is not supported.

Code example for accessing standard Java classes:

```
function main()

{

      var flow = new FlowContext- Class;

      /*Create an instance of class string which is part of java.lang
package. */

      var myString = new java.lang.String();

      myString.format("Hello World");

      /* Get the length of the string.*/

      var length = myString.length();

      flow.SetResult(0);

}
```

SENDCONTACTCENTERDATA

The SendContactCenterData block sends the memo constructed in the JScriptExecute block to the agent.

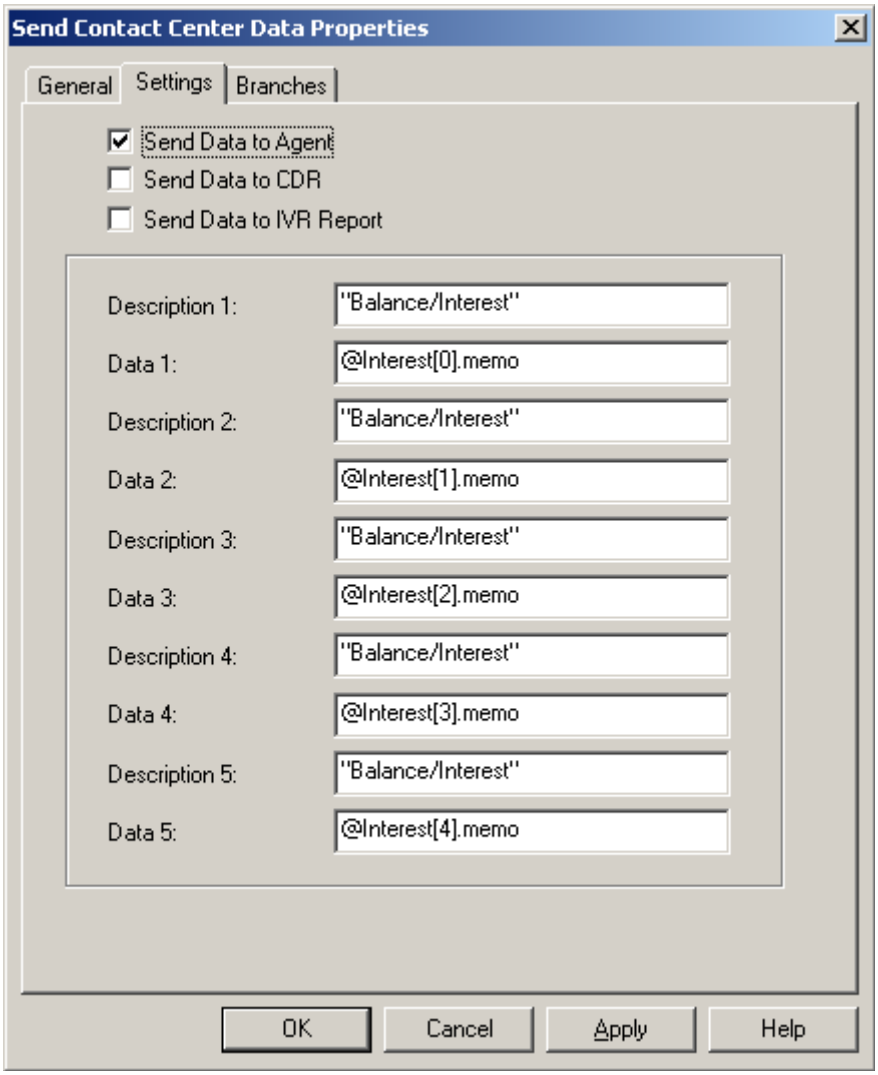


Figure 50: SendContactCenterData settings

SERVICE GROUP

The Service Group block defines to which Service Group the call will be sent.

MICONTACT CENTER AGENT SESSIONS WINDOW

Once the call is routed to an available agent, the constructed memo will be displayed in the MiContact Center Agent Sessions window. The same information can also be sent to CDR logging by enabling the Send Data to CDR option in the SendContactCenterData block.

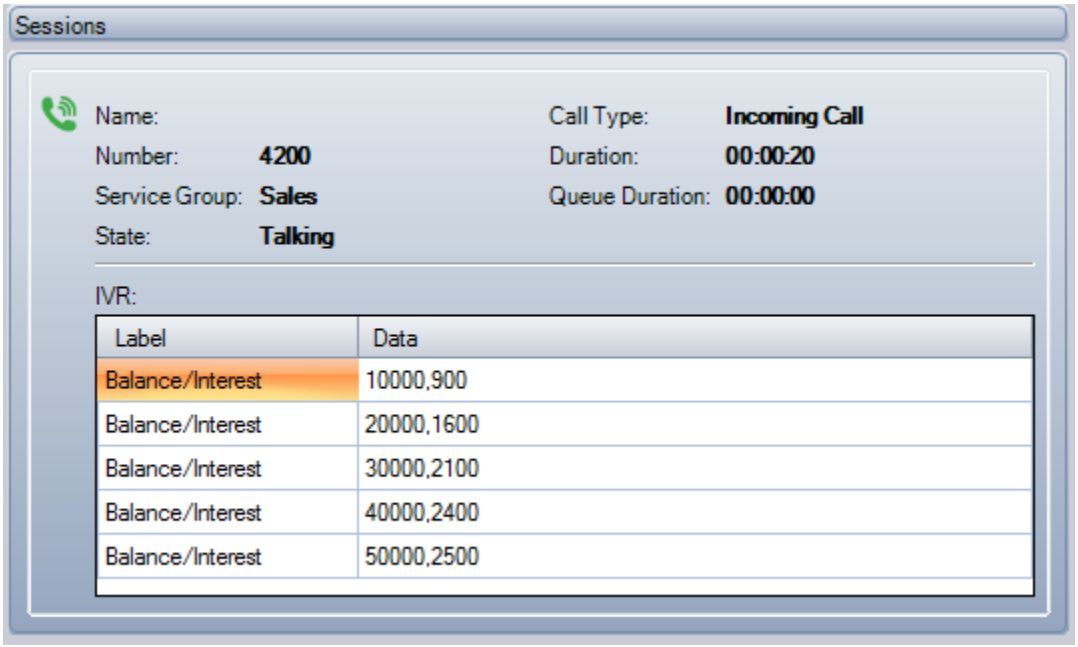


Figure 51: MiContact Center Agent Sessions Window

TESTING THE APPLICATION

Make sure the Service Group name is configured in Configuration Manager. Create a Service Access using compiled script via Configuration Manager. Use SpyTracer as a debugging tool to see the execution of the script.

VBSCRIPT EXECUTE

The VBScriptExecute Component in Script Manager has full integration with Windows. This component relies on the Windows Script Host for its VBScript functionality. This will open opportunities for communication with outside processes and web servers.

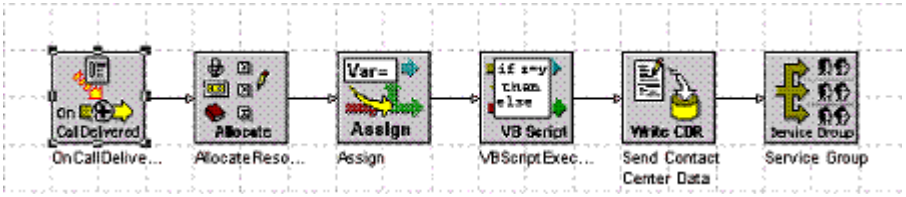
Running a process outside Script Manager and waiting for the result can be easily done using the VBScript component.

This tutorial demonstrates the use of the VBScriptExecute and the Object data type. The VBScriptExecute block accesses the variable defined in the script, manipulates the data and formats it to a string. The text string is then sent to MiContact Center Agent.

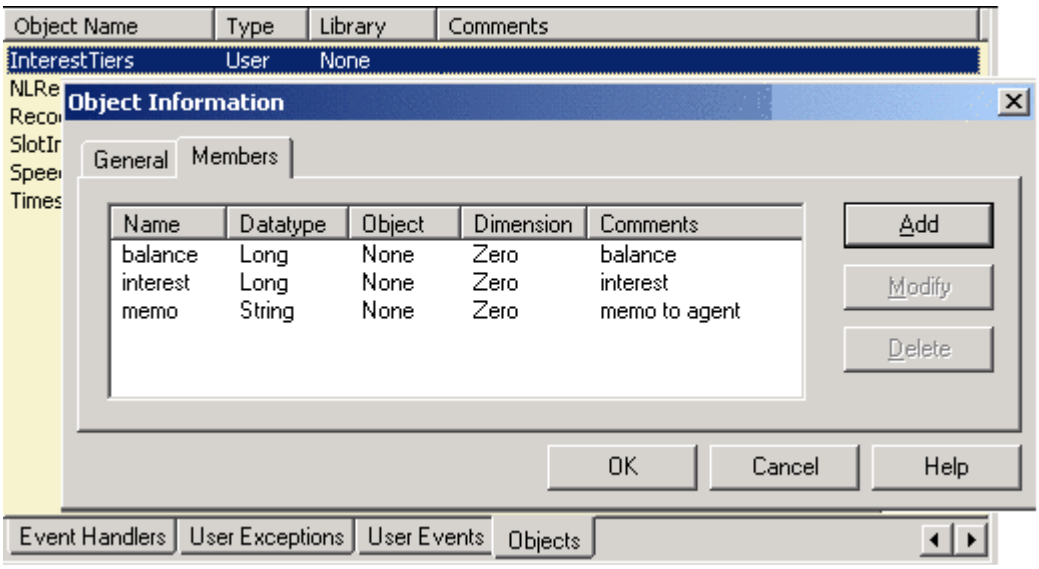
VBSCRIPT.MFD

To use the VBScript block, do the following:

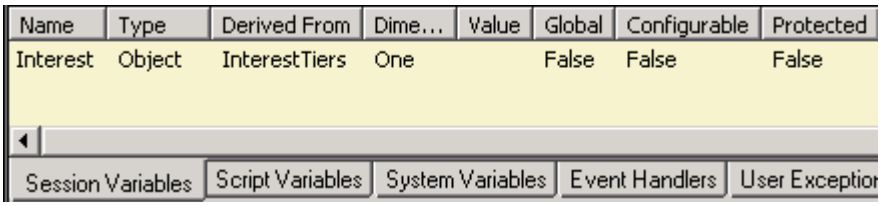
1. Open the tutorials project from Script Designer, and click on VBScript to open the script.



2. Select the **Objects** tab from Script Object Pane. A user defined object called **InterestTiers** is defined.
3. Double click on **InterestTiers** and the **Object Information** dialog displays the members defined. In this example, there are three members defined: balance, interest and memo.



4. Click on the **Sessions Variables** tab. A variable called **Interest** is defined as a one dimensional array of the **InterestTiers** data type.



5. After receiving an incoming call the script makes use of the balances settings from the **Assign** block.

ASSIGN

The **Assign** block sets the balance of each element in the @Interest array to a different value.

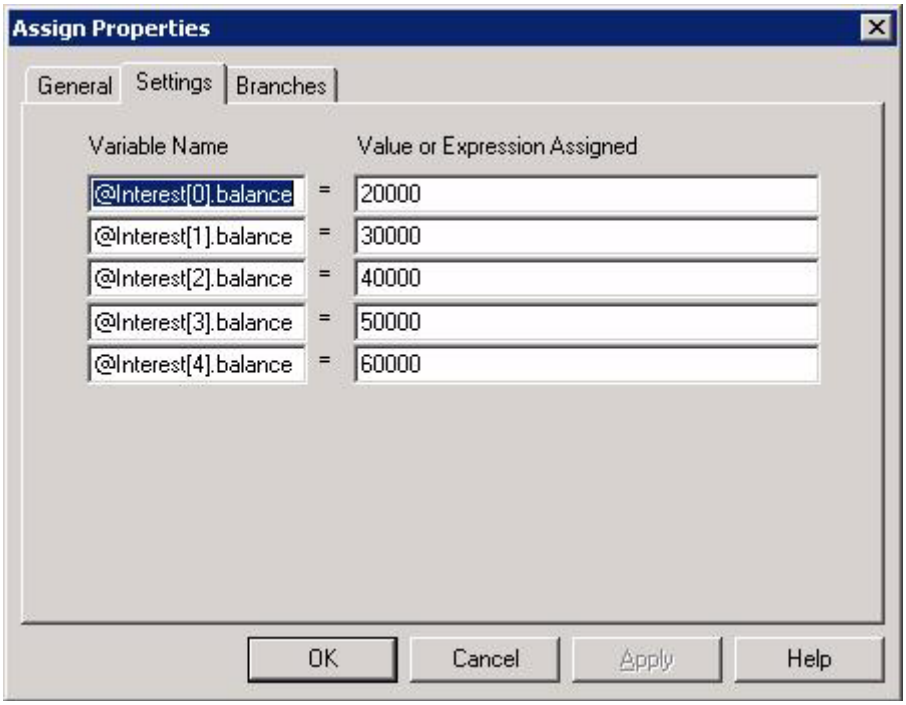


Figure 52: Assigning values to variables

Note The first element of the array has index 0. This value will be used in the VBScriptExecute block to calculate different interests.

VBSCRIPTEXECUTE

The settings tab of the VBScriptExecute Properties is presented in the figure below.

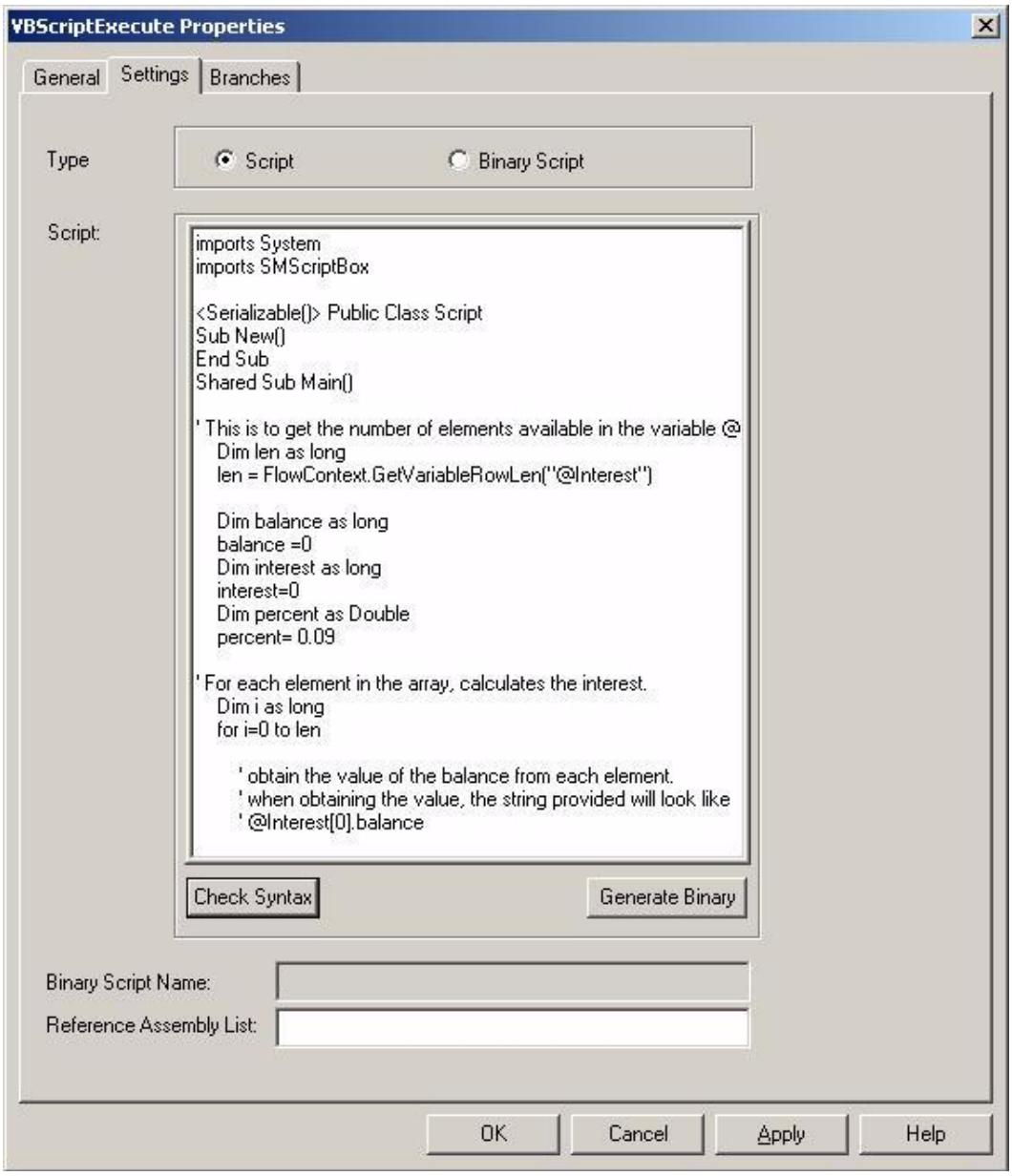


Figure 53: Settings tab of VBScriptExecute Properties

Note: If the Reference Assembly List field is used to specify dependent assemblies for the script, they must be accessible to the compiled script which by default is located in the Windows directory. Alternatively, the dependent assemblies can be signed with a strong name and added to the Global Assembly Cache with GacUtil.

```
'This is to get the number of elements available in the
variable @Interest

Dim len as long

len = FlowContext.GetVariable- RowLen(“@Interest”)

Dim balance as long

balance =0

Dim interest as long

interest=0

Dim percent as Double

percent= 0.09

' For each element in the array, calculates the interest.

Dim i as long

for i=0 to len

     ' obtain the value of the balance from each element

     ' when obtaining the value, the string provided will look
like

     ' @Interest[0].balance

     balance = FlowContext.GetVariable(“@Interest
[“+i.toString()+”].balance”

     interest = balance * percent

     percent = percent - 0.01

     ' save the calculated interest to the
@Interest[i].interest variable
```

```
     FlowContext.SetVariable("@Interest["+i.toString()+"].inte
rest", interest)

     ' now prepare the data to present to the agent

     Dim memo as String

          memo = balance.toString() + ","+ interest.toString()

          ' save the data to the @Interest[i].memo

          FlowContext.SetVariable("@Interest["+i.toString()+"]
          .memo", memo)

Next

' set the result to 0

FlowContext.SetResult(0)
```

SENDCONTACTCENTERDATA

This block sends the memo constructed in the VBScriptExecute block to the agent.

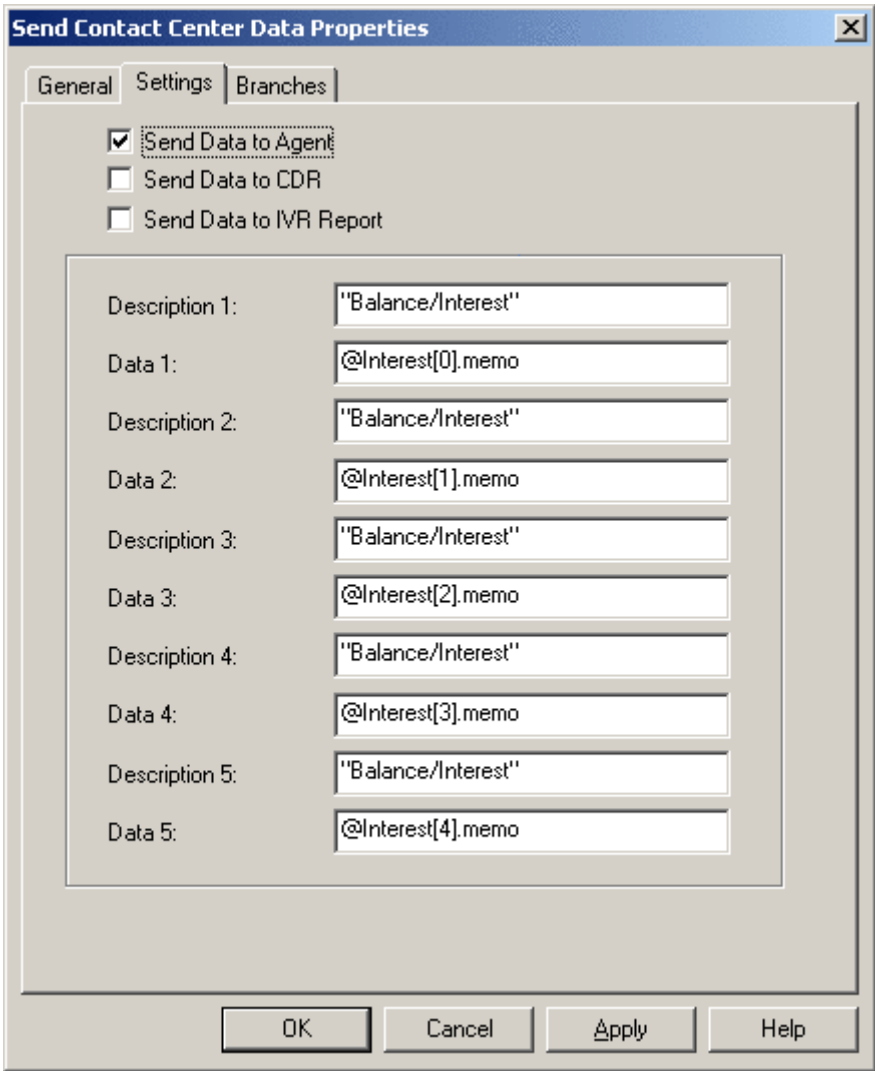


Figure 54: Settings tab of Send Contact Center Data

SERVICE GROUP

The Service Group block defines to which Service Group the call will be sent.

MICONTACT CENTER AGENT SESSIONS WINDOW

Once the call is routed to an available agent, the constructed memo will be displayed in the MiContact Center Agent Sessions window. The same information can also be sent to CDR by enabling **Send Data to CDR** option in the SendContactCenterData block.

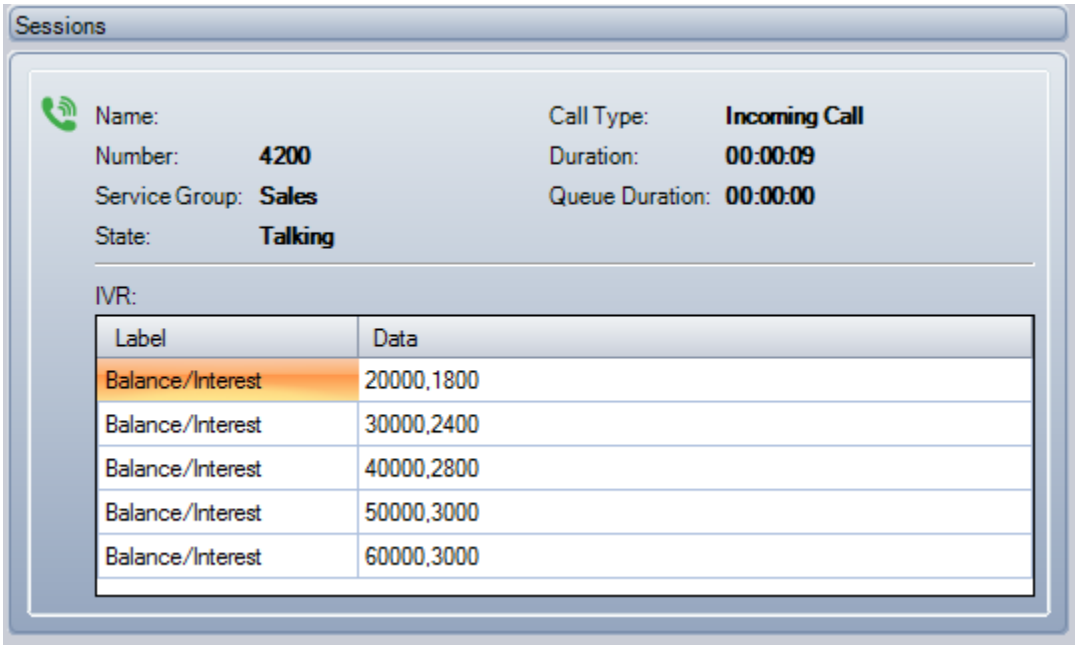


Figure 55: MiContact Center Agent Sessions Window

TESTING THE APPLICATION

Make sure the Service Group name is configured in Configuration Manager. Create a Service Access using the compiled script via Configuration Manager. Use SpyTracer as a debugging tool to see the execution of the script. For more information on Spy Tracer, please see document *Debugging Applications*.

VBSCRIPT COMMUNICATION WITH WEB SERVER EXAMPLE

This example shows how the communication abilities of the VBScript component can be used.

VBScript component will connect to a Web Server and retrieve an XML file thru HTTP and GET methods. Later, it will save the XML file locally on the hard drive and parse through the file to find out the number to call. This number will be passed to the Script Manager DeflectCall component and the call will be deflected to that number.

In this example the deflected to number is an extension in use by MiContact Center Agent.

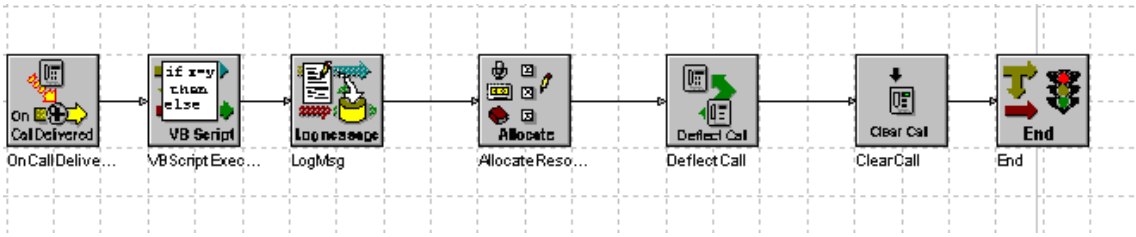


Figure 56: VBScript communication with web server script

VBScript script has been implemented as shown below:

```
imports System

imports Microsoft.VisualBasic.Interaction

imports SMScriptBox

<Serializable()> Public Class Script

Shared Sub Main()

' Query http server and use GET to retrieve a file

Const ForWriting = 2

Dim strURL

Dim WshShell

strURL="http://localhost/IMSSetting.xml"

Dim objHTTP

objHTTP = CreateObject("MSXML2.ServerXMLHTTP")

Call objHTTP.Open("GET", strURL, FALSE)

objHTTP.Send

' Save the file on local hard drive.

Dim objFSO
```

```
Dim objFile

objFSO = CreateObject("Scripting.FileSystemObject")

objFile = objFSO.CreateTextFile("C:\local.xml",ForWriting)

objFile.Write(objHTTP.ResponseText)

objFile.Close

' Parse XML file

Dim xmlDoc, objNodeList, plot, x

xmlDoc = CreateObject("Msxml2.DOMDocument")

xmlDoc.load("c:\local.xml")

objNodeList = xmlDoc.getElementsByTagName("PublicUserId")

If objNodeList.length > 0 then

      For each x in objNodeList

            Plot = x.Text

            Next

      Else

            FlowContext.SetResult(1)

      End If

      ' Pass the result to Script Manager

      FlowContext.SetVariable("@SipUser", plot)

      FlowContext.SetResult(0)

      End Sub

      End Class
```

Notes on the script:

- It is possible to import the base classes inside the VBScript component.
- Parentheses should be used in function calls, even if in some function calls in common VBScript they can be ignored.
- Result should be set by the script to Script Manager FlowContext. If the Result is ignored, then the script will return failure.
- A server process such as the SM script engine (FlowProcessor.exe) must use a "ServerXMLHTTP" object to send HTTP requests. (The “XMLHTTP” object is only valid for a client process.