Mitel MiContact Center Enterprise

DATABASE COMPONENTS APPLICATIONS SCRIPT MANAGER USER GUIDE

Release 9.1

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.

No part of this document can be reproduced or transmitted in any form or by any means - electronic or mechanical - for any purpose without written permission from Mitel Networks Corporation.

TRADEMARKS

The trademarks, service marks, logos and graphics (collectively "Trademarks") appearing on Mitel's Internet sites or in its publications are registered and unregistered trademarks of Mitel Networks Corporation (MNC) or its subsidiaries (collectively "Mitel") or others. Use of the Trademarks is prohibited without the express consent from Mitel. Please contact our legal department at legal@mitel.com for additional information. For a list of the worldwide Mitel Networks Corporation registered trademarks, please refer to the website: http://www.mitel.com/trademarks.

Database Components Applications Script Manager
User Guide
Release 9.1 – May 2016

®,™ Trademark of Mitel Networks Corporation
© Copyright 2016 Mitel Networks Corporation
All rights reserve

INTRODUCTION

This document contains sample scripts using components provided in the database component library. Some components in the call and media control library are also covered in this document. The tutorial provides a set of sample scripts and useful hints when building the script.

WHAT YOU WILL LEARN

In this document the following topics are presented in detail:

1. How to build scripts to access the ODBC compliant database using the Database Components.
2. How to open and close the connection to the data source.
3. How to prepare SQL statements using SQL query or stored procedures.
4. How to bind and fetch data, and execute the prepared SQL statements.

BEFORE USING THE TUTORIAL

Make sure the sample scripts are installed on your development PC. Refer to Installing Sample Scripts for instructions if not already installed.

The tutorial requires an ODBC compliant database to access and run the sample customer script. If you have not already done so, run the SampleCustomer.bat to create the sample database called “SampleCustomerDB”. SampleCustomer.bat and its associated files are located in the <InstallDir>\ MiCC Enterprise\Script Manager\SampleScripts\SampleDBScript directory. The following parameters are required to run the sampleCustomer.bat:

- Path for the setup directory.
- Password for the super user “sa”. It is empty in this example.
- The SQL Server name
- The full path and name where the database device "samplecustomerlg.mdf" file is to reside
- name where the database log "samplecustomerlg.ldf" file is to reside
- Debug log info

For example, enter the following command in the DOS prompt. Make sure the directory for the database storage is created before using this command.

SampleCustomer "C:\Program Files\Mitel\MiCC Enterprise\ScriptManager\SampleScripts\ SampleDBScript" "sapassword" bt-solidus "c:\MiCC Enterprise\SampleData\ samplecustomerlg.mdf" ”c:\MiCC Enterprise\SampleData\samplecustomerlg.ldf" "debug"

Note: By default, most SA passwords are null or empty. However if the SQL Server SA account was created with a password, then the password word for SA should be entered for the command parameter “sapassword” above.

Otherwise, just use “” to represent the empty string for the password parameter.

Once the database is successfully created, follow the steps below to create an ODBC data source from the control panel.

1. Use the Administrative Tools from the Control Panel, and select Data Source (ODBC). This opens the ODBC Data Source Administrator properties.
2. Select System DSN tab and select Add.
3. Select a driver, for example SQL Server, and click Finish.
4. Enter a name for the data source, for example SampleCustomer.
5. Enter a description, for example sample customer database for Script Manager.
6. Select the SQL Server name, and click Next.
7. Select With SQL Server authentication using login ID....
8. Enter the login ID and the password to access the SampleCustomerDB. By default, “sa” is the authorized user with the permission to access and login. Enter the “sa” user password, the default is empty. Click Next.
9. Check Change the default database to and select SampleCustomerDB as the database. Click Next and Finish to complete the data source configuration.
10. Test the data source and make sure the connection can be made.

Note: To configure Data Sources (ODBC) on a Windows 64-bit system, you need to use “C:\Windows\SysWOW64\ODBCAD32.exe”.

TUTORIALS

In this section, tutorials for ValidateCustomer, ValidateSTPCustomer using Stored Procedure and Customer Banking using Statement and State are described.

VALIDATECUSTOMER

This tutorial demonstrates the usage of how ValidateCustomer sub-script is invoked from the main-script CallingValidateCustomer. How to create a calling sub-script from the main script can be found in Basic Call Handling and Advanced Applications.

VALIDATECUSTOMER.SFD.

This sub-script will focus on the usage of the Database Components by opening the OBDC data source connection to prepare and execute the SQL statement with the binding input and output parameters, for an overview of the use of the sub-script see Figure 1.

From Script Designer, open the project called tutorials.fdp from the SampleScripts directory. This opens the tutorial project workspace. Double click on the CallingValidateCustomer to open the main script, and see how the ValidateCustomer sub-script is called.

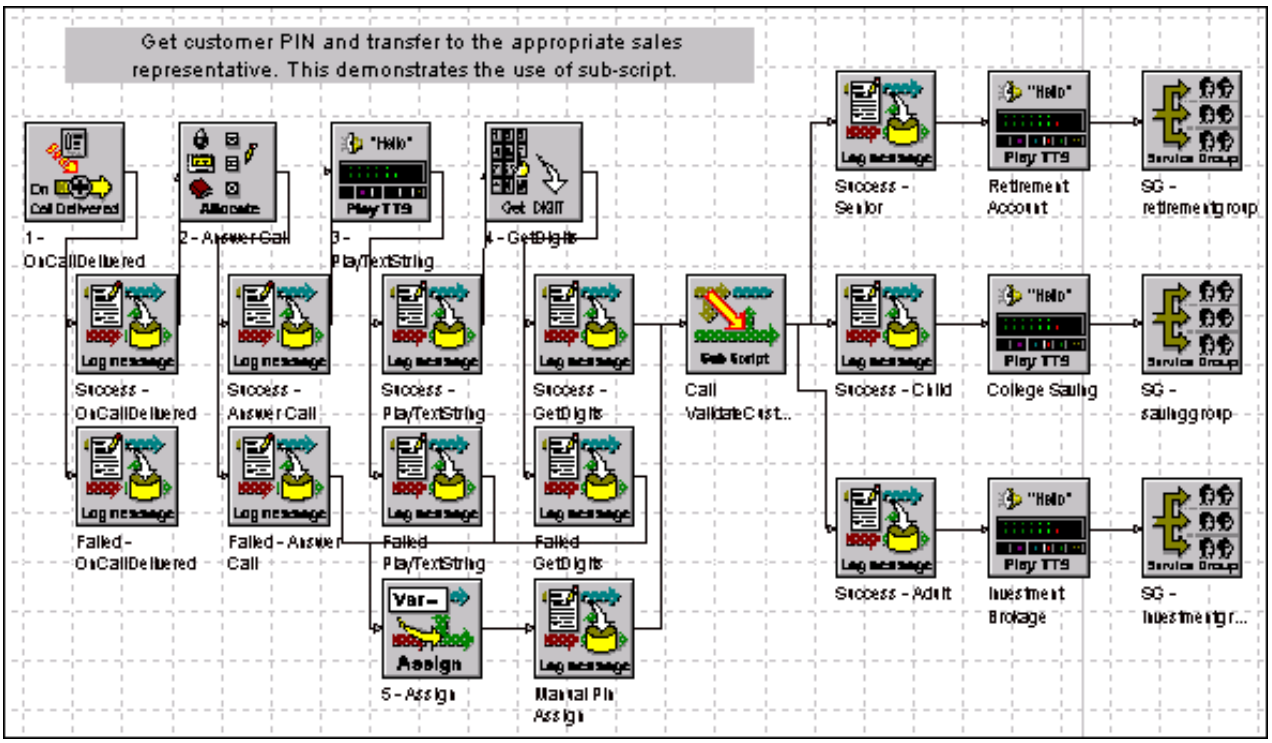

Figure 1: Validate Customer sub-script

The components details in the CallingValidateCustomer main script will not be discussed in this document. The CallingValidateCustomer will handle an incoming call of the bank customers. It allocates a Text-to-Speech resource and prompts the customer to enter a 4 digit customer pin. Once the digits have been received, it will feed these digits into the “ValidateCustomer” sub-script. The sub-script will query for the customer Age and Name so the call can be routed to the appropriate service group to serve this customer.

From the tutorials.fdp project workspace, double click on the ValidateCustomer to open the sub-script.

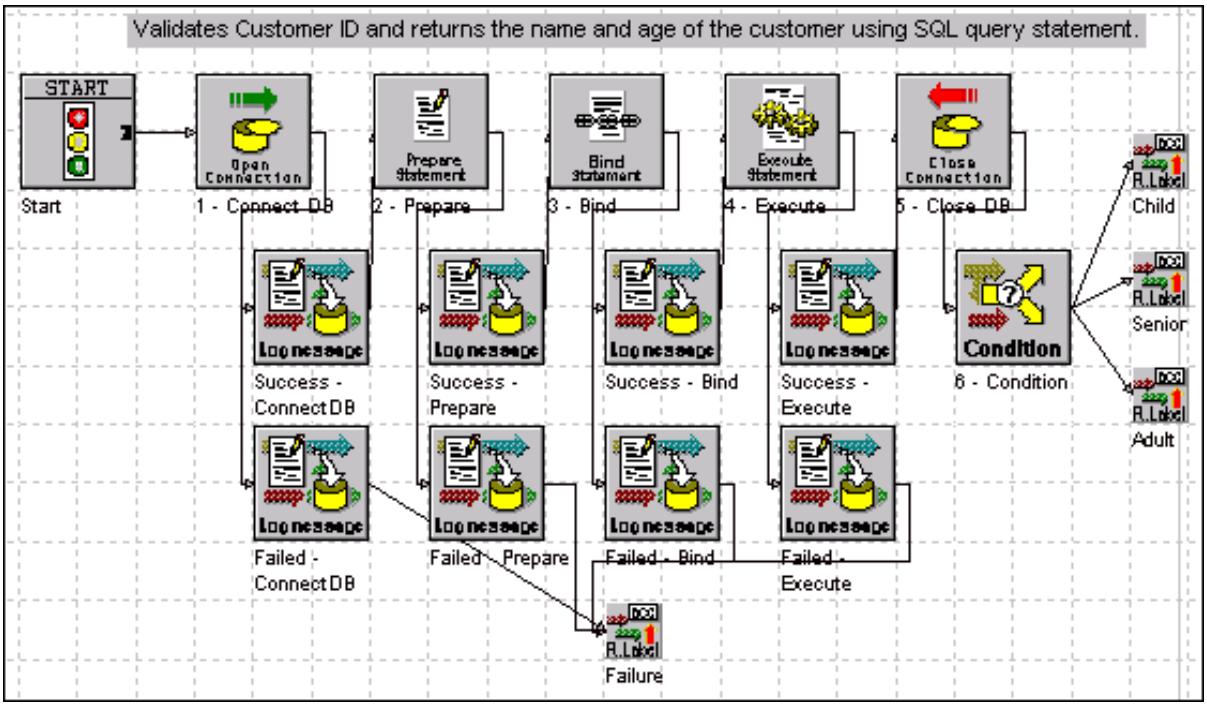

Figure 2

The purpose of the ValidateCustomer script is to connect to the data source named “SampleCustomer” for customer data and query information. In the example, the name and age of the customer will be queried based on the input Pin number from the CallingValidateCustomer main-script.

When the ValidateCustomer sub-script is called, it connects to the OpenConnection block. The OpenConnection block is responsible for establishing a connection to the specified OBDC data source that will be used throughout in this script. View the properties of the OpenConnection block.

OPEN CONNECTION

In the OpenConnection block setting properties, see Figure 3SampleCustomer is the data source name, which will be used in the Validate Pin script. Username and Password fields are the account information of the user that has permission to access this SampleCustomer database.

“sa” is used in this sample with no password required. The Time-out field is the number of seconds to wait for a connection.

If the Use Cursor Library check box is enabled, the ODBC cursor library will be loaded. The default option is Use driver, which provides the scrolling capabilities of the driver. Other available options include: Use if needed and Use ODBC. The Use if needed uses the cursor library only if needed when selected. And selecting the Use ODBC option will set the driver manager to use the cursor library.

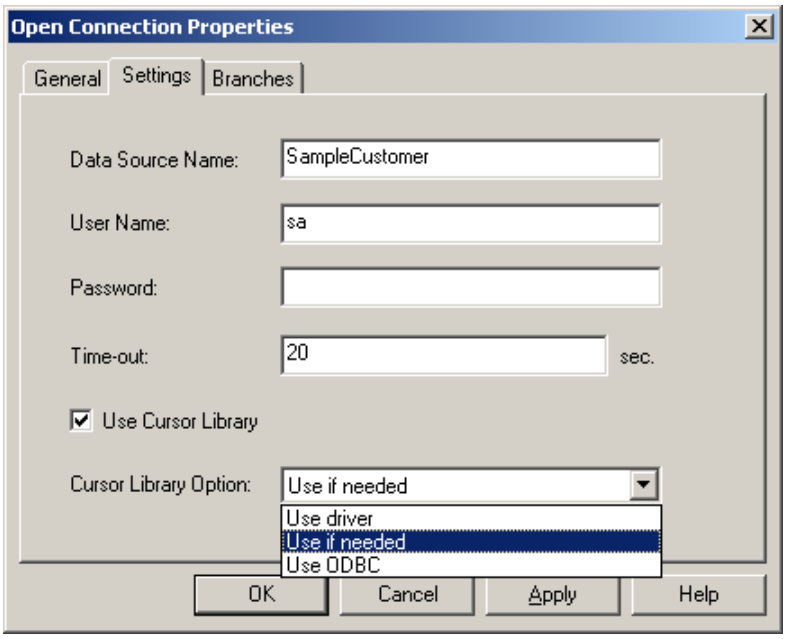

Figure 3: Settings tab of Open Connection Properties

PREPARE

Once the connection to the database has been successfully established, the Prepare block is used to prepare the SQL statement for a later execution using the Execute block. On this Prepare block Settings, Data Source Name will be the same “SampleCustomer” ODBC data source, see Figure 4.

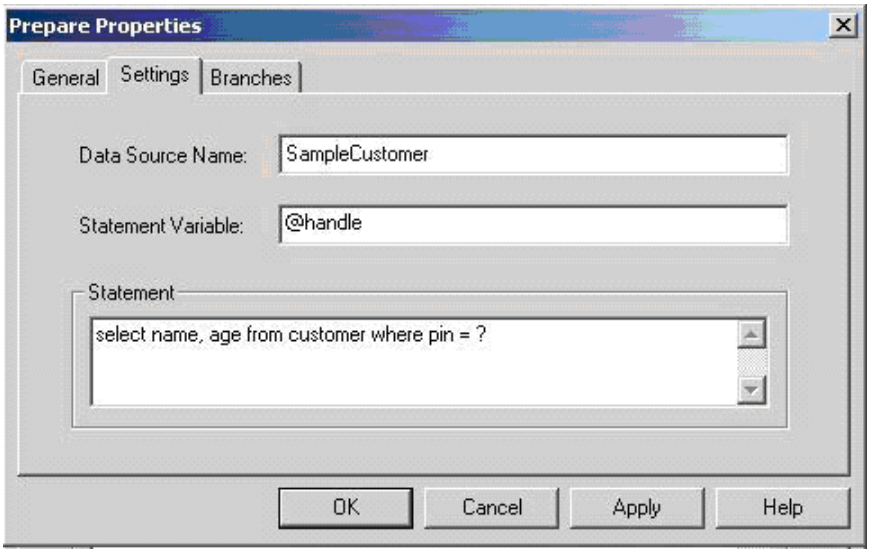

Figure 4 Settings tab of Prepare Properties

Defining the input and output variables of the sub-script

The Statement Variable @handle is defined as a Session Variable as shown in Figure 5. In the ValidateCustomer script, four variables are defined as Session Variables. Since the variables will only be used in the Startup sub-script session, and not in the Event-Driven or Shutdown sessions, they are not defined as Script Variables. Refer to the Online Help for more detailed information about Variables.

To define a variable, go to the Script Object Pane and switch to the Session Variables tab.

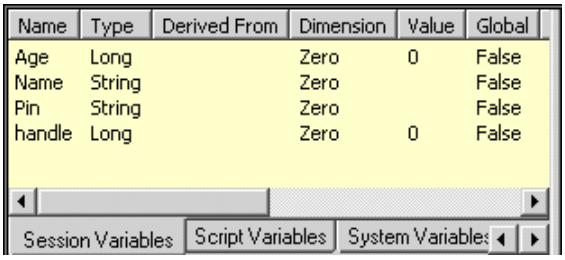

Figure 5: Session Variables

This ValidateCustomer script requires the Pin variable as the input parameter and returns the retrieved data to the Name and Age variables. The handle variable is used to store the returned handle from the OpenConnection block, which the Prepare, Bind, and Execute blocks will use.

The last section of the Prepare block is the Statement section, which contains the SQL statement for later execution in the Execution block. The SQL statement “select name, age from customer where pin = ?” defined in this Prepare block requires an input Pin value indicated as “?”. The @Pin variable, which has the customer entered value from GetDigit block in the calling script, will be used as an input parameter to this SQL. The Name and Age data will then be retrieved from the Customer table once the prepared statement is executed with the valid input Pin.

Input parameter and Passing the input data to the sub-script

The Input Parameters tab, see Figure 6, defines the input parameter to the sub-script. There are three ways to pass input data to the sub-script: input parameters, variable mapping and global variables. In the “ValidateCustomer” sample script, the input parameters method is used.

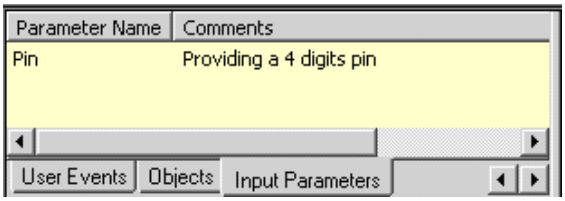

Figure 6: Input Parameters tab

BIND

Data can be transferred between the application and the ODBC compliant database. Data must be bound before it can be transferred. The Bind block, see Figure 7, binds the data as variables to parameters in the same order as the parameters are specified in the prepared SQL statement. Variable data can be bound either via the Statement/Procedure and/or the Output Column parameters.

In the Bind block, the same data source name (SampleCustomer) and statement handle (@handle) from the Prepare block are used to bind the variable data.

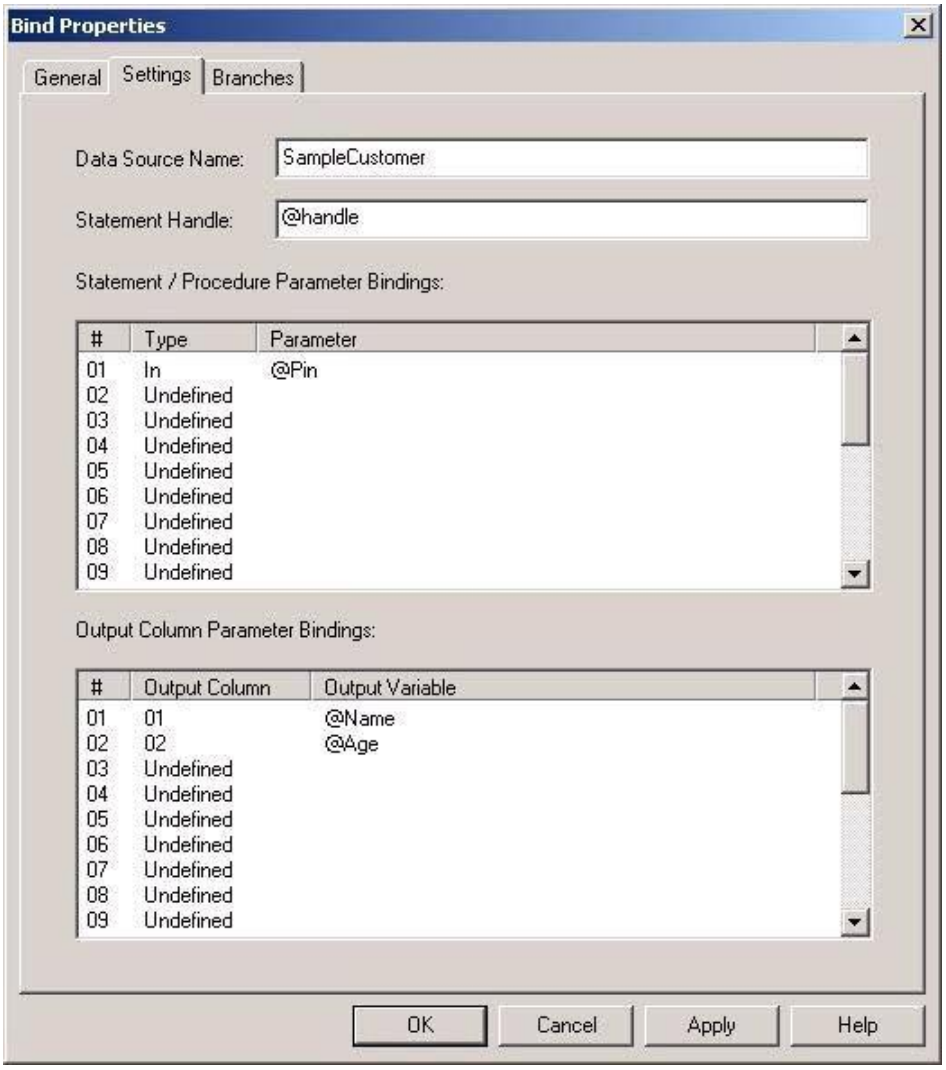

Figure 7: Settings tab of Bind Properties

Binding Statement / Procedure Parameters

Double-click on the first row (#1) to open the Data Type. The Statement / Procedure Parameter Bindings dialogue box will appear. Enter the name of the parameter (@Pin variable is used in this example). Specify the Parameter Type from the drop-down list. Parameter names will be dependent on the Parameter Type chosen:

1. In - Can be either a Variable or a Constant
2. In-Out - Variable
3. Out - Variable
4. Return - Variable
5. Undefined - Used for removing data from the selected parameter and all parameters that follow.

In this example, the Customer Pin will be the input to the prepared SQL statement. Therefore, the variable @Pin is selected as an Input type for binding. The @Pin variable can be selected to bind as an In-Out type also. For example, if a customer needs to have a Pin number change, then the customer would expect to provide the old Pin and have a new Pin returned as well. When the @Pin variable binds as an In-Out type, it is used as an input variable and an output variable type as well.

Binding Column Parameters

From the ValidateCustomer script, there are 2 variables @Name and @Age expected to be returned as a result from the @Pin input, therefore @Name and @Age variables need to be bound as Output Parameters. Based on the prepared SQL statement of “select name, age from customer where pin = ?”, the order of expected output variables from “select ....” statement has name first and then age. The parameters need to have the variable @Name bound as the first output column #1 and @Age as the #2 column.

To define a variable for output binding, double-click on the first open Output Column. The Output Column Parameter Bindings dialogue box will appear. Specify the Output Column from the drop-down list and enter the name of the Output Variable. The Bind command will bind the variables to the columns in the same order as the variables are specified in the prepared SQL statement.

EXECUTE

Once the bound variables have been defined as input and output parameters, the prepared SQL statement can then be executed via the Execute block, see Figure 8.

The Execute Statement block executes the SQL statement previously prepared in the Prepare block. If the Fetch First Record option is selected, it will attempt to fetch the first record in this block (if the statement executed is expected to return bound column data as in the ValidateCustomer script).

In the Execute block, the Data Source Name is the name of the ODBC data source; “SampleCustomer”. The Statement Handle is the @handle variable from the data source connection.

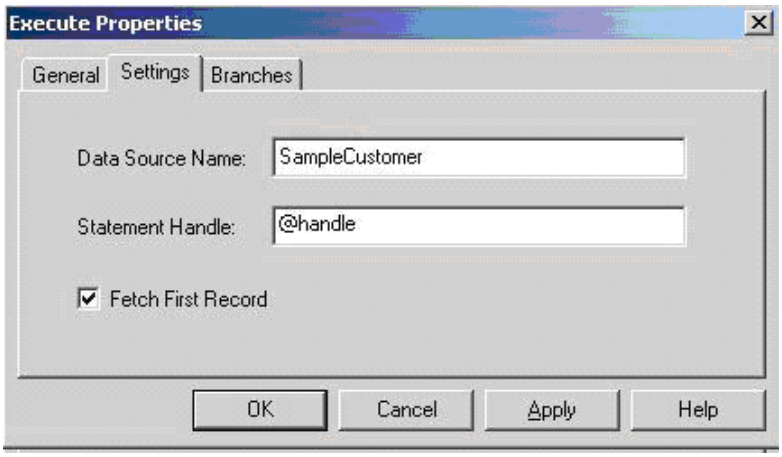

Figure 8: Settings tab of Execute Properties

CLOSE CONNECTION

The Close Connection block closes any previously opened OBDC database connection before returning to the calling script.

Once the customer's @Name and @Age information has been retrieved, the ValidateCustomer sub-script will return to the calling script with a specific return label based on the age group of the customer. In the CallingValidateCustomer script, if the returned label is a child, it plays a text message for the child customer, and the call will be routed to the Service Group that handles the college saving. Similar conditions apply to the adult or senior age group customers.

In Summary, the ValidateCustomer script opens a connection to the SampleCustomer ODBC data source (Open Connection block). A valid @handle will be available on a successful connection. A prepared SQL Statement is defined to query the Customer's Name and Age using the Customer's Pin (Prepare block). Before the data can be fed into or retrieved from the ODBC database, variables must be defined and bound as In, In-Out and/or Output Parameters for the prepared SQL statement (Bind block).

The prepared SQL statement can then be executed in the Execute block. A successful execution will return the Customer's Name and Age stored in variables, @Name and @Age. The connection to the SampleCustomer data source must be closed before returning to the calling script to prevent resource leaks and a dangling connection.

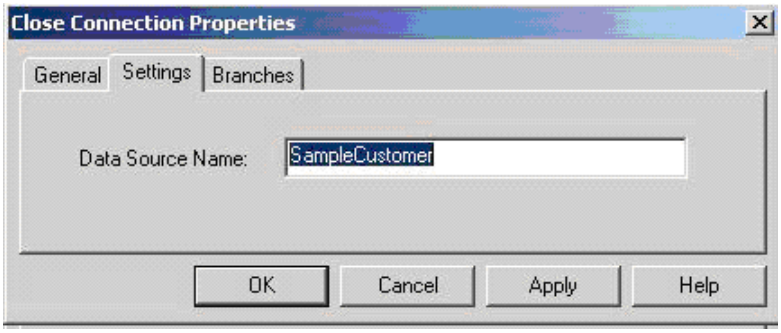

Figure 9

VALIDATESTPCUSTOMER USING STORED PROCEDURE

The previous example demonstrated that data can be retrieved from the database using a simple prepared SQL statement. Another method to obtain the same information is to execute a stored procedure.

A stored procedure is a precompiled executable object that contains one or more SQL statements. Stored procedures can have zero or more input and output parameters and can issue an integer return code. Parameters passed into a stored procedure can be hard-coded or variable. Use a parameter marker (?) to specify the parameters. The block binds a program variable to the parameter marker, and then places the data value in the program variable.

Executing a stored procedure is similar to executing a prepared statement, except that the stored procedure exists as a permanently compiled object in the database. A stored procedure can also be used to hide complex SQL statements from the application.

VALIDATESTPCUSTOMER.SFD

In this sample, instead of using a standard SQL query statement in the Prepare block as in the ValidateCustomer sub-script, the ValidateSTPCustomer uses a predefined stored procedure for retrieving customer data.

From the Script Designer tutorials.fdp project, double click on the ValidateSTPCustomer to open the sub-script.

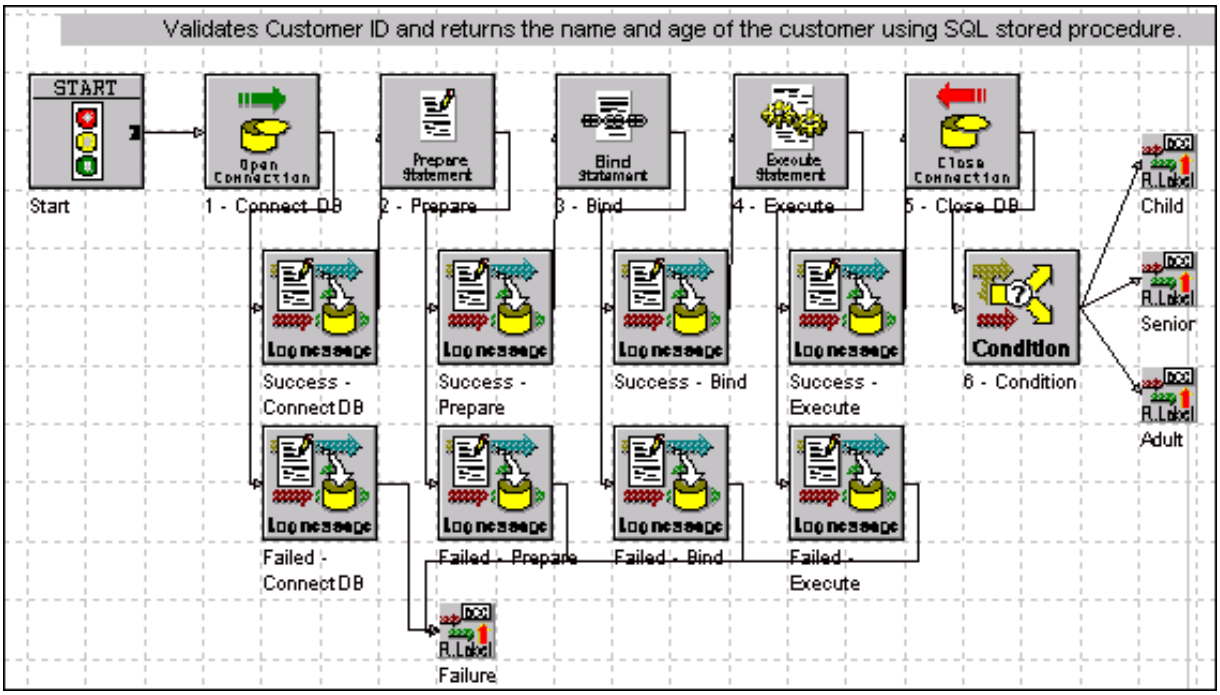

Figure 10

The ValidateSTPCustomer sub-script is invoked by the CallingSTPValidateCustomer main-script. The calling script is similar to the CallingValidateCustomer with exception of calling ValidateSTPCustomer in the sub-script block.

In the ValidateSTPCustomer sub-script, all ODBC components being used here are similar to the previous example with the exception that the Stored Procedure is prepared in the Prepare block. In the Prepare block, “stp_getcustomer” is the name of the stored procedure to be executed.

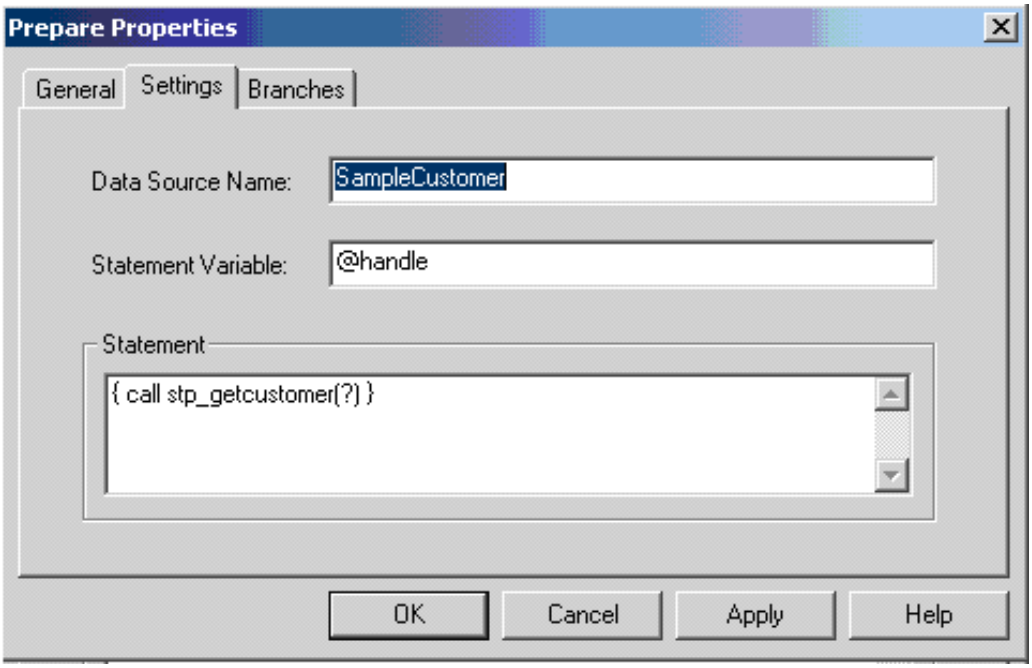

Figure 11 Settings tab of Prepare Properties

The stored procedure call statement is shown below.

{ call stp_getcustomer(?) }

Note: There may be differences on how to execute a stored procedure for different databases. Try “Execute stp_getcusomer (?)” if the above does not work properly.

Like the SQL query “select name, age from customer where pin = ?” from the Prepare block of the ValidateCustomer sub-script, the “stp_getcustomer” stored procedure will perform the similar query when executed. To pass the data to the stored procedure and read the data back, @Pin will be passed as (?) to the stored procedure “stp_getcustomer”, and the output variables @Name and @Age will contain the data once the stored procedure is executed. In the Bind block, @Pin is bound as input (?) and @Name and @Age are bound as output.

The stored procedure will be executed in the ValidateSTPCustomer sub-script. The stored procedure is created when the SampleCustomerDB is created using the SampleCustomer.bat. It can be located in the Stored Procedure folder under the SampleCustomer database. The “stp_getcustomer” stored procedure performs a SQL query “select @CustName = name, @CustAge = age from Customer where pin = @CustPin”. It is similar to the prepared SQL statement in ValidateCustomer example where the name and age from the customer table will be assigned back to the stored procedure output parameters @CustName and @CustAge.

```
set nocount on

GO

CREATE procedure stp_getcustomer @CustPin char(255)

as set nocount on

declare @CustName char(255)

declare @CustAge int

select @CustName = name, @CustAge = age from Customer where pin =

@CustPin

return

GO
```

There are 2 other sample stored procedures that are created when the SampleCustomerDB is created. They are “stp_getcustomerName” and “stp_getcustomerAge”, which works in a similar manner to “stp_getcustomer”. The “stp_getcustomerName” stored procedure is used to query the Customer Name and “stp_getcustomerAge” is to query the Customer Age from the input Pin.

CUSTOMER BANKING USING STATEMENT AND STATE

There are advanced ODBC components that can be used to create more powerful applications.

In this tutorial, approaches for data access with different states and option settings in the SQL query statement is demonstrated. The “Customerbanking” script from this tutorial will demonstrate a more complex call handling scenario using these ODBC components.

CUSTOMERBANKING.MFD

From the project workspace, double click on the “Customerbanking” to open the script.

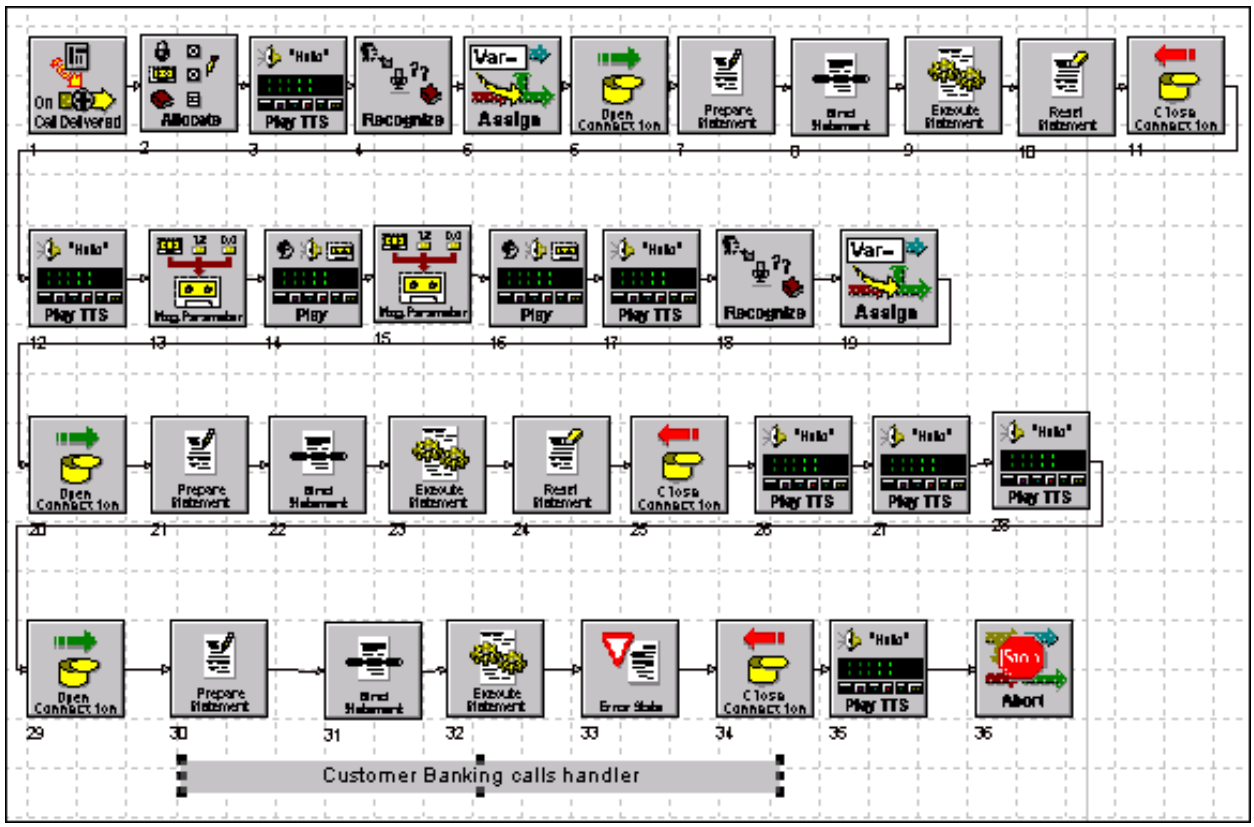

Figure 12

In the Customerbanking script, when an incoming call from a bank customer is answered, a voice recognition, recorder and player are allocated. The customers ID will be prompted using Text-to-speech. The Recognize component waits for the customer to speak and tries to recognize the spoken words. Once the customer ID has been recognized, it will assign the ID to the@id variable, which will be used as an input parameter to the prepared SQL statement. At this time, the data source connection to the “SampleCustomer” database will be opened. The “Prepare” block #7 will prepare the “select name, balance, e-mail from Customer where id = ?” SQL statement with @id variable as the input parameter. Once this prepared SQL statement is executed, the customer Name, balance and e-mail will be queried from the database with the matching customer id.

After the Execute block is executed, the statement is reset using the “Reset Statement” block. The purpose of this Reset Statement is to end the statement processing by the first “Prepare” block, #7, and close the associated cursor.

It also discards the pending result that is associated with the statement. This reset statement is necessary when the script needs to prepare and execute more than one statement. In the “Reset Statement” block, the Statement Handle is the statement to be reset. In this example, the handle is @first.

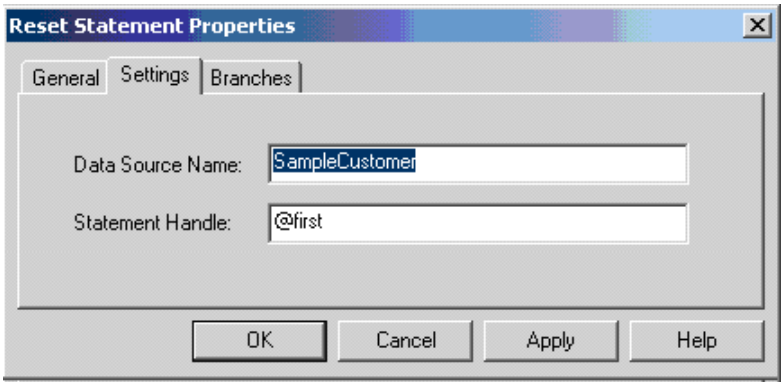

Figure 13

After the statement is reset, the connection to the database can be closed to prevent any resource leakage.

The Customerbanking script will then read the retrieved data and play back the data with banking information such as name and current to the customer

Then another prompt will be played to ask the customer to provide the new updated name and ID to update the customer account. At this time, the statement @second handle will reset to close any associated cursor or pending result. The Recognize block, #18, now waits for the customer's answers to identify for name and ID. The new updated name and ID will be input as input parameters for the prepared statement "update new set id = ? where name = ?" in the Prepare block, #21. Customer will then hear a play message of the successful update ID once the prepared statement is executed successfully and an e-mail notification is sent out.

To prepare for the e-mail to be sent out, the Prepare block prepares the SQL statement using a predefined system stored procedure called "xp_sende-mail" in the Master database. The customer @e-mail and new @update ID have previously been retrieved and remain in the variables, therefore they are used as input into the "xp_sende-mail" stored procedure.

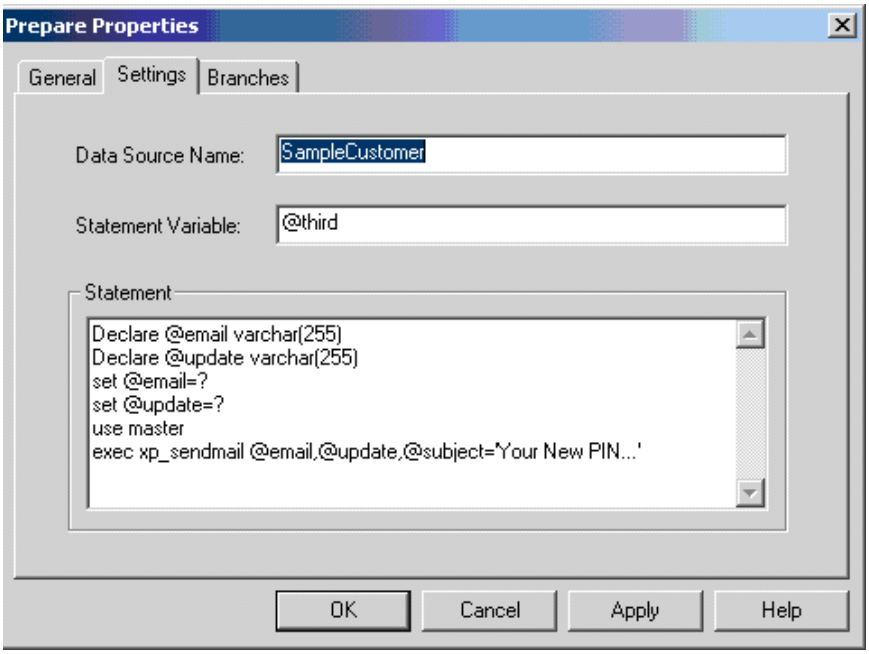

Figure 14

When this stored procedure is executed, an e-mail will be sent to the customer with the subject headline as “Your New PIN”.

After the “xp_sende-mail” is executed, the Statement State block is used to define the branch conditions based on the state set by ODBC.

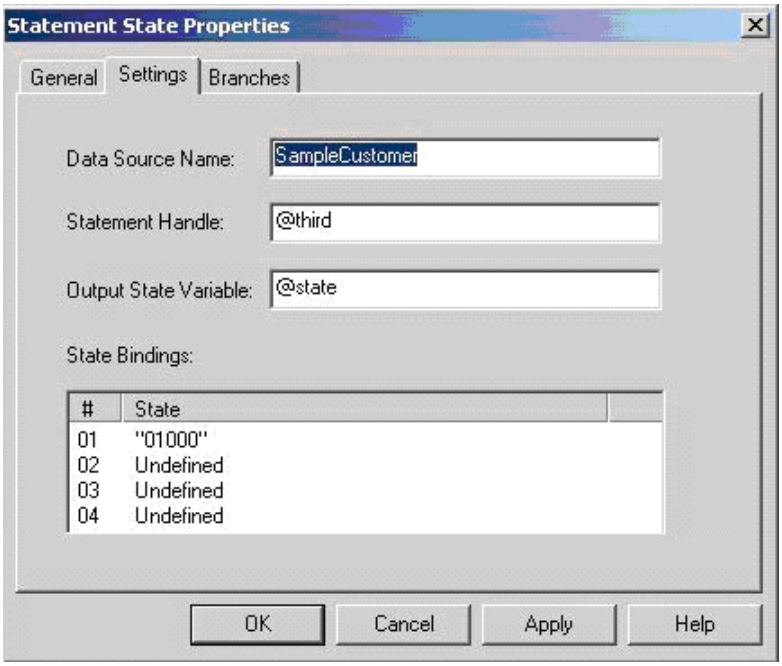

Figure 15

In the Statement State block, the SampleCustomer and @third variables are used from the third data source connection. The @state variable is used to store the returned statement state set by ODBC. The State Bindings section defines the branch it will take when matches the result state.

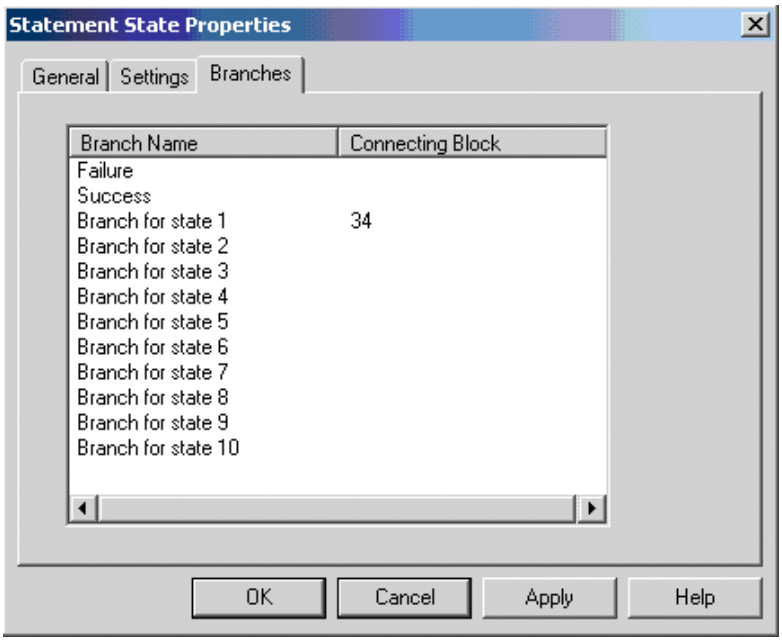

Figure 16

In the Customerbanking script, the only defined branch is that if the returned state is “01000”, or successful, then it should branch to block #34 to close the database connection. Finally a message will be played to the customer indicating that an e-mail has been sent before ending the call.

Mitel
Powering connections

mitel.com

© Copyright 2014, Mitel Networks Corporation. All Rights Reserved. The Mitel word and logo are trademarks of Mitel Networks Corporation. Any reference to third party trademarks are for reference only and Mitel makes no representation of ownership of these marks.