

VXscript IVR Example

```
Uses Call, DB, Speech;

Static WarningsLeft;

#####

## DisconnectCall: Helper function that clears the call and stops script
## processing
##
## Parameters: none
## Return value: never returns
##

Function DisconnectCall {
Call::ImmediateAction = Call::actionDisconnect;

Halt;
}

#####

## ErrorHandler: Database error handler function
##
## Parameters: none
## Return value: never returns
##
## This function plays a recording to the user indicating that a database
## error has occurred and disconnects the call.
##

Function ErrorHandler {
Call::PlayFile("DatabaseError.wav");

DisconnectCall();
}

#####

## EmptyDigitBuffer: Helper function that clears the digit collection buffer
##
```

```

## Parameters: none

## Return value: none

##

Function EmptyDigitBuffer {

Call::GetDigits(20, 1, 1);

}

#####

## ForceCents: Round a monetary value up to the next cent

##

## Parameters: What - monetary value

## Return value: the monetary value rounded up to the next cent

##

Function ForceCents(What) {

Return Int(What * 100 + 0.5) / 100;

}

#####

## PlayPersonalGreeting: Play the personal greeting for this account, if it

## has one

##

## Parameters: none

## Return value: 0 if the personal greeting file did not exist

## anything other than 0 if it did

##

Function PlayPersonalGreeting {

Return Call::PlayFile("PersonalGreetings\\" . Call::AccountIdentifier .

".wav", "#");

}

#####

## CheckCallingNumber: Attempt to determine the user's account code from

## their caller ID information

##

## Parameters: none

## Return value: 1 if successful

```

```

## 0 if not successful

##

## The account code will be stored in Call::AccountIdentifier, if
## successful.

##

Function CheckCallingNumber {
SetErrorHandler(ErrorHandler);

Var AccountCode = DB::TranslateCallingNumber(Call::IncomingTrunkGroup,
Call::CallingNumber);

If (DB::Status == DB::dbsSuccess) {
Call::AccountIdentifier = AccountCode;

Return 1;
} Else {
Return 0;
}
}

#####

## GetPIN: Prompt the user to enter their PIN, up to three times

##

## Parameters: none

## Return value: the PIN the user entered, or "" (empty string) if none was
## entered

##

Function GetPIN {
Var Tries;
Var PIN;

Do {
Call::PlayFile("EnterPIN.wav", "0123456789#");

PIN = Call::GetDigits(20, 2000, 5000, "#");

If (" " != PIN) {
Return PIN;
}

Tries++;
}
}

```

```

} While (Tries < 3);

Return "";

}

#####

## GetAccountCode: prompt the user to enter their account code, up to three
## times, followed by their PIN, if needed
##
## Parameters: none
## Return value: none
##
## The account code will be stored in Call::AccountIdentifier, if
## successful. This function does not return if the user does not enter a
## valid account code and PIN within three tries.
##
Function GetAccountCode {
SetErrorHandler(ErrorHandler);

Var Tries;
Var AccountCode;
Var PIN;

Do {
#
# Prompt the user to enter their account code, and collect the
# digits.
#
Call::PlayFile("EnterAccountCode.wav", "0123456789#");
AccountCode = Call::GetDigits(20, 2000, 5000, "#", 0);
If (AccountCode != "") {
#
# The user dialed a number. Check that it's a valid account code.
#
DB::ValidateAccountID(Call::IncomingTrunkGroup, AccountCode);
If (DB::Status == DB::dbsSuccessNoPIN) {

```

```

# It is, and the user's account has no PIN. Save the account
# code in the Call module, so it will be available to other
# parts of this script and also recorded in CDRs. Return
# 1 (for success).

Call::AccountIdentifier = AccountCode;

Return;

} Else If (DB::Status == DB::dbsNeedPIN) {

# The account exists and is valid, but we need to collect
# the PIN from the user and verify it.

PIN = GetPIN();

If (PIN != "") {

# The user entered a PIN - now verify it

DB::ValidateAccountID(Call::IncomingTrunkGroup,
AccountCode, PIN);

If (DB::Status == DB::dbsSuccess ||
DB::Status == DB::dbsSuccessNoPIN) {

# Save the user's account code in the Call module, so
# it will be recorded in any CDRs, etc., as above.

Call::AccountIdentifier = AccountCode;

Return;

} Else If (DB::Status == DB::dbsAccountExpired) {

Call::PlayFile("YourAccountHasExpired.wav");

Call::PlayFile("HaveANiceDay.wav");

DisconnectCall();

} Else If (DB::Status == DB::dbsAccountDisabled) {

Call::PlayFile("YourAccountHasBeenDisabled.wav");

Call::PlayFile("HaveANiceDay.wav");

DisconnectCall();

} Else If (DB::Status == DB::dbsAccountOverdrawn) {

Call::PlayFile("YourAccountIsOverdrawn.wav");

Call::PlayFile("HaveANiceDay.wav");

DisconnectCall();

} Else {

# The user did not enter the correct PIN - try
again.

# We will play the "try again" recording to the user

```

```

# below.
}

} Else {
# The user did not enter a PIN - try again.
# We will play the "try again" recording to the user
below.
}
} Else {
# No such account - try again.
# We will play the "try again" recording to the user below.
}
Call::PlayFile("TryAccountAgain.wav", "0123456789");
}
Tries++;
} While (Tries < 3);

# No account code was entered, so hang up.
Call::PlayFile("Goodbye.wav");
DisconnectCall();
}

#####
## GetNumberToCall: prompt the user to enter number they wish to call, up to
## three times
##
## Parameters: none
## Return value: the number the user dialed, if they dialed one, or ""
## (empty string) if the user does not enter a number.
##
Function GetNumberToCall {
Var Tries;
Var Digits;
Do {
Call::PlayFile("DialNumber.wav", "1234567890#");
Digits = Call::GetDigits(20, 5000, 10000, "#");
If (Digits != "") {

```

```

Return Digits;
}

Tries++;
} While (Tries < 3);

Return "";
}

#####
## PlaceCall: Place a call for the user
##
## We verify that the user has enough available credit to be able to place
## a call at least as long as the first billing interval.
##
## Parameters: none
## Return value: Only returns if the user cannot place this call or does not
## enter a phone number.
##
## The call will be placed directly from this function if the user enters
## a number and can afford to place the call.
##
Function PlaceCall {
Var MaxCallLength;
Var NumberToCall;

# Ask the user for the number to call
NumberToCall = GetNumberToCall();

If (NumberToCall == "") {
Return;
}

#
# Retrieve the user's account balance and limit, the rate for the number
# they wish to call, and the first and second billing intervals from the
# database, and use this information to calculate the longest amount of

```

```

# time the user may place the call for before running out of money.
#

Var AccountBalance = DB::GetAccountBalance(Call::IncomingTrunkGroup,
Call::AccountIdentifier);
IF (DB::dbsSuccess != DB::Status) {
Call::PlayFile("UnableToGetRate.wav");
Return;
}

Var AccountLimit = DB::GetAccountLimit(Call::IncomingTrunkGroup,
Call::AccountIdentifier);
If (DB::dbsSuccess != DB::Status) {
If (DB::dbsAccountUnlimited != DB::Status) {
Call::PlayFile("UnableToGetRate.wav");
Return;
}
# The user's account has no credit limit - place the call and
# permit them to talk for as long as they like.
Call::ImmediateAction = Call::actionPlaceCall;
Call::ClearBackAction = Call::actionRecallScript;
Call::ClearForwardAction= Call::actionRecallScript;
Call::TimerExpiryAction = Call::actionDoNothing;
Call::NextActionTime = 0; # never
Call::CalledNumber = NumberToCall;
Halt;
}

Var AvailableMoney = AccountLimit - AccountBalance;

# The rate is measured in currency units (dollars, for example) per
# minute.
Var Rate = DB::GetRate(Call::IncomingTrunkGroup, NumberToCall);
If (DB::dbsSuccess != DB::Status) {
Call::PlayFile("UnableToGetRate.wav");
Return;
}

```



```

if (Rate == 0) {
# This call is free! - place the call and
# permit them to talk for as long as they like.
Call::ImmediateAction = Call::actionPlaceCall;
Call::ClearBackAction = Call::actionRecallScript;
Call::ClearForwardAction= Call::actionRecallScript;
Call::TimerExpiryAction = Call::actionDoNothing;
Call::NextActionTime = 0; # never
Call::CalledNumber = NumberToCall;
Halt;
}

# The first billing interval is the shortest possible call length,
# measured in seconds.
Var FirstBillingInterval =
DB::GetFirstBillingInterval(Call::IncomingTrunkGroup,
NumberToCall);
If (DB::dbsSuccess != DB::Status) {
Call::PlayFile("UnableToGetRate.wav");
Return;
}
Var FirstBillingIntervalCost = FirstBillingInterval / 60 * Rate;

# After the first billing interval, the call is billed for in increments
# measured by the second billing interval, in seconds.
Var SecondBillingInterval =
DB::GetNextBillingInterval(Call::IncomingTrunkGroup,
NumberToCall);
If (DB::dbsSuccess != DB::Status) {
Call::PlayFile("UnableToGetRate.wav");
Return;
}
Var SecondBillingIntervalCost = SecondBillingInterval / 60 * Rate;

# Tell the user how much this call will cost per minute.

```

```

EmptyDigitBuffer();

Call::PlayFile("CallRate.wav", "1#");

Speech::SayMoney(Rate, "1#");

Call::PlayFile("PerMinute.wav", "1#");

# Ensure that the user can actually afford to place a call for the minimum
# duration (that is, the first billing interval).

If (FirstBillingIntervalCost > AvailableMoney) {
Call::PlayFile("NotEnoughMoney.wav");

If (AvailableMoney >= 0) {
Call::PlayFile("YouOnlyHave.wav");

Speech::SayMoney(AvailableMoney, "");

Call::PlayFile("Available.wav");

} Else {
Call::PlayFile("AccountOverdrawnBy.wav");

Call::PlayFile("HaveANiceDay.wav");

Speech::SayMoney(AvailableMoney, "");

}

EmptyDigitBuffer();

Return;

}

# Deduct the cost of the minimum call length from the local variable
# representing the user's available money.

AvailableMoney = AvailableMoney - FirstBillingIntervalCost;

# The user has enough funds to place a call for at least the minimum
# length of time. Now calculate how much more time they can afford.

Var AdditionalCallLength =
Int(AvailableMoney / SecondBillingIntervalCost) *
SecondBillingInterval;

MaxCallLength = FirstBillingInterval + AdditionalCallLength;

Call::PlayFile("MaximumCallLength.wav", "1#");

Speech::SayTimeRemaining(MaxCallLength, "1#");

```

```

Call::PlayFile("CancelCall.wav", "1#");
Var Cancel = Call::GetDigits(1, 2000, 5000);
If (Cancel == 1) {
# The user changed their mind -- don't actually place the call.
Return;
}

Call::ImmediateAction = Call::actionPlaceCall;
Call::ClearBackAction = Call::actionRecallScript;
Call::TimerExpiryAction = Call::ActionRecallScript;
Call::NextActionTime = MaxCallLength - 30;
Call::CalledNumber = NumberToCall;
WarningsLeft = 2;
Halt;
}

#####
## ChangePersonalGreeting: Record a new personal greeting for this account
##
## Parameters: none
## Return value: none
##
Function ChangePersonalGreeting {
# Instruct the user in how to record their greeting
Call::PlayFile("Record.wav");

# Record the greeting
Call::RecordFile("PersonalGreetings\\" . Call::AccountIdentifier . ".wav",
300, "#");

# Now play back their recorded greeting. We clear the digit buffer
# first to prevent accidental cancellation.
EmptyDigitBuffer();
Call::PlayFile("RecordedGreetingIs.wav", "#");
PlayPersonalGreeting();

```

```

# Discard the "#" the user may have dialed to cancel the playback
Call::GetDigits(1, 1, 1, "#");
}

#####

## MainMenu: Perform the main menu

## Option 1: place a call

## Option 2: change personal greeting

## Option "#": hang up

##

## Parameters: none

## Return value: 1 if the user wishes to place a call

## 2 if the user wishes to change their personal greeting

## If the user does not enter a choice after three prompts,

## or if the user requests to hang up, this function does

## not return.

##

Function MainMenu {

Var Tries;

Var Option;

Do {

Call::PlayFile("MainMenu.wav", "12#");

Option = Call::GetDigits(1, 2000, 5000);

If (Option == "1") { # Place a call

Return 1;

} Else If (Option == "2") { # Change personal greeting

Return 2;

} Else If (Option == "#") {

Call::PlayFile("Goodbye.wav");

DisconnectCall();

} Else {

Call::PlayFile("InvalidChoice.wav");

Tries++;

}
}

```

```

} While (Tries < 3);

# No choice was made, so disconnect the call.
Call::PlayFile("Goodbye.wav");
DisconnectCall();
}

#####
## HandleNewCall: Handle a new incoming call
## * Greet the user
## * Ask the user for their account code
## * Ask the user for their PIN, if necessary
## * Tell the user what their account's balance is
##
## Parameters: none
## Return value: none
##
Function HandleNewCall {
SetErrorHandler(ErrorHandler);

Var Balance;
Var AccountLimit;

# Initialize this call's release reason to "normal clearing".
Call::ReleaseReason = 16;

# Wait for one second, then connect the call.
Call::PlayFile("1SecondPause.wav");
Call::Connect();

# First, let's try to guess the user's account code by looking at their
# caller ID information
If (CheckCallingNumber() == 0) {
# We couldn't, so we now need to prompt the user for their account
# code (and their PIN, if necessary). Permit the greeting to be
# interrupted by dialing any digit.

```

```

Call::PlayFile("Greeting.wav", "0123456789");

GetAccountCode();

# Play the user's personal greeting, if it exists

PlayPersonalGreeting();

} Else {

# We succeeded in determining the user's account code from their caller
# ID, so try to play their personal greeting. If they don't have one,
# PlayPersonalGreeting() will return 0, and we'll play the standard
# greeting instead.

If (PlayPersonalGreeting() == 0) {

Call::PlayFile("Greeting.wav", "#");

}

}

# If the user interrupted the greeting by pressing #, we need to
# remove the '#' from the digit buffer.

Call::GetDigits(1, 1, 1, "#");

# Retrieve the user's account balance from the database and tell them
# what it is. Say nothing if they have a zero balance.

Balance = ForceCents(DB::GetAccountBalance(Call::IncomingTrunkGroup,
Call::AccountIdentifier));

If (DB::Status != DB::dbsSuccess) {

Call::PlayFile("DatabaseError.wav");

DisconnectCall();

}

If (Balance != 0.00) {

Call::PlayFile("YourAccountBalanceIs.wav", "1234567890#");

Speech::SayMoney(Balance, "1234567890#");

If (Balance < 0) {

Call::PlayFile("Credit.wav", "1234567890#");

}

}

# Retrieve the user's credit limit from the database and tell them
# what it is.

AccountLimit = ForceCents(DB::GetAccountLimit(Call::IncomingTrunkGroup,

```

```

Call::AccountIdentifier));

If (DB::dbsSuccess != DB::Status) {

If (DB::dbsAccountUnlimited != DB::Status) {

Call::PlayFile("DatabaseError.wav");

DisconnectCall();

}

Call::PlayFile("YourAvailableBalanceIs.wav", "1234567890#");

Call::PlayFile("Unlimited.wav", "1234567890#");

} Else {

Var AvailableMoney = AccountLimit - Balance;

Call::PlayFile("YourAvailableBalanceIs.wav", "1234567890#");

Speech::SayMoney(AvailableMoney, "1234567890#");

If (AvailableMoney < 0) {

Call::PlayFile("OverDrawn.wav", "1234567890#");

Call::PlayFile("HaveANiceDay.wav");

DisconnectCall();

}

}

}

#####

## HandleClearBack: Handle a call that is returning to the script because

## the party the user called disconnected.

##

## Inspect Call::ReleaseReason to see why the called party disconnected

## and inform the user of some common problems (number busy, etc.), then

## ask them if they wish to place another call or return to the main menu.

##

## Parameters: none

## Return value: none

##

Function HandleClearBack {

# Examine Call::ReleaseReason, and tell the user why their call

# disconnected

If (Call::ReleaseReason == 17) { # subscriber busy

```

```

Call::PlayFile("Busy.wav", "12");

} Else If (Call::ReleaseReason == 3) { # no route to destination

Call::PlayFile("NoRoute.wav", "12");

} Else If (Call::ReleaseReason == 28) { # invalid number

Call::PlayFile("InvalidNumber.wav", "12");

} Else If (Call::ReleaseReason == 16) {

# Normal clearing - don't actually say something

} Else {

Call::PlayFile("UnknownReleaseReason.wav", "12");

Speech::SayNumber(Call::ReleaseReason);

}

# Re-set the release reason, as before (in HandleNewCall())

Call::ReleaseReason = 16;

# Find out what the user wishes to do now

Var Tries;

Var Option;

Do {

Call::PlayFile("ReDial.wav", "12");

Option = Call::GetDigits(1, 2000, 5000);

If (Option == 1) { # Place another call

PlaceCall();

} Else If (Option == 2) { # Return to the main menu

Return;

}

Tries++;

} While (Tries < 3);

# No choice was made, so disconnect the call.

Call::PlayFile("Goodbye.wav");

DisconnectCall();

}

#####

## HandleTimerExpired: Handle a call that is returning to the script

```



```

## because the NextActionTimer expired

##

## Looks at the WarningsLeft global variable to decide whether to
## disconnect the call, or warn the user that they have a short amount of
## time left.

##

## Parameters: none

## Return value: none

##

Function HandleTimerExpired {

If (WarningsLeft >= 1) {

# Tell the user how many seconds they have left, with 15 seconds
# between warnings

Call::PlayFile("YouHave.wav");

Speech::SayTimeRemaining(WarningsLeft * 15);

Call::PlayFile("Remaining.wav");

# Set up the call to return to the script

Call::ImmediateAction = Call::actionContinueCall;

Call::ClearBackAction = Call::actionRecallScript;

Call::TimerExpiryAction = Call::actionRecallScript;

Call::ClearForwardAction= Call::actionRecallScript;

Call::NextActionTime = 15; # 15 seconds from now

WarningsLeft--;

Halt;

} Else {

# No more warnings - hang up the call now.

Call::PlayFile("Goodbye.wav");

DisconnectCall();

}

}

#####

## Main: The primary entry point into this script

##

## This function will be invoked directly by the IVR system inside VX,

```

```

## and should be the function configured as the AAA function on a channel.

##

## Parameters: none

## Return value: none

##

Function Main {
Call::LogCDR=1;

debug("### Script Invocation reason is " . Call::InvocationReason);

debug("### Warnings left is " . WarningsLeft);

#call::connect();

WarningsLeft=99;

If (Call::InvocationReason == Call::reasonIncomingCall) {
HandleNewCall();
} Else If (Call::InvocationReason == Call::reasonClearBack) {
# The call is returning to the script after the outgoing call
# the user placed has disconnected
HandleClearBack();
} Else If (Call::InvocationReason == Call::reasonTimerExpiry) {
# The call is returning to the IVR because the NextActionTimer has
# expired
HandleTimerExpired();
} Else If (Call::InvocationReason == Call::reasonClearForward) {
# The call is returning to the IVR after the user himself
# has disconnected
Debug("*****User disconnected.***** Callback occurred!");
Debug("Reading a call timer - Connect : " . Call::ConnectTime);
Debug("Reading a call timer - Disconnect: " . Call::DisconnectTime);

if (call::connecttime > 0) {
Debug("Call Length: " . (Call::DisconnectTime-Call::ConnectTime)/10000000 . " seconds");
}

Halt;
}

```

```
# Clear out the digit buffer so that the user cannot inadvertently
# select a main menu option
EmptyDigitBuffer();

Var Option;

Do {
# Find out what the user wishes to do
Option = MainMenu();

If (Option == 1) { # Dial a number
PlaceCall();
}

If (Option == 2) { # Change recorded greeting
ChangePersonalGreeting();
}

} While (Option > 0);

# Option will never be 0 -- MainMenu() will exit the script directly
# via Halt if the user stops responding.
```

}