Using the Script Editor

Last Updated: July 24, 2008

This chapter describes how to use the Cisco Unity Express Script Editor in the following sections:

- Overview of the Cisco Unity Express Script Editor, page 9
- Palette Pane, page 10
- Design Pane, page 11
- Variable Pane, page 12
- Debug Pane, page 18
- Using Prompts, page 18
- Using Expressions, page 18
- Using Operators, page 19
- Handling Exceptions, page 20
- Installing the Cisco Unity Express Script Editor, page 23

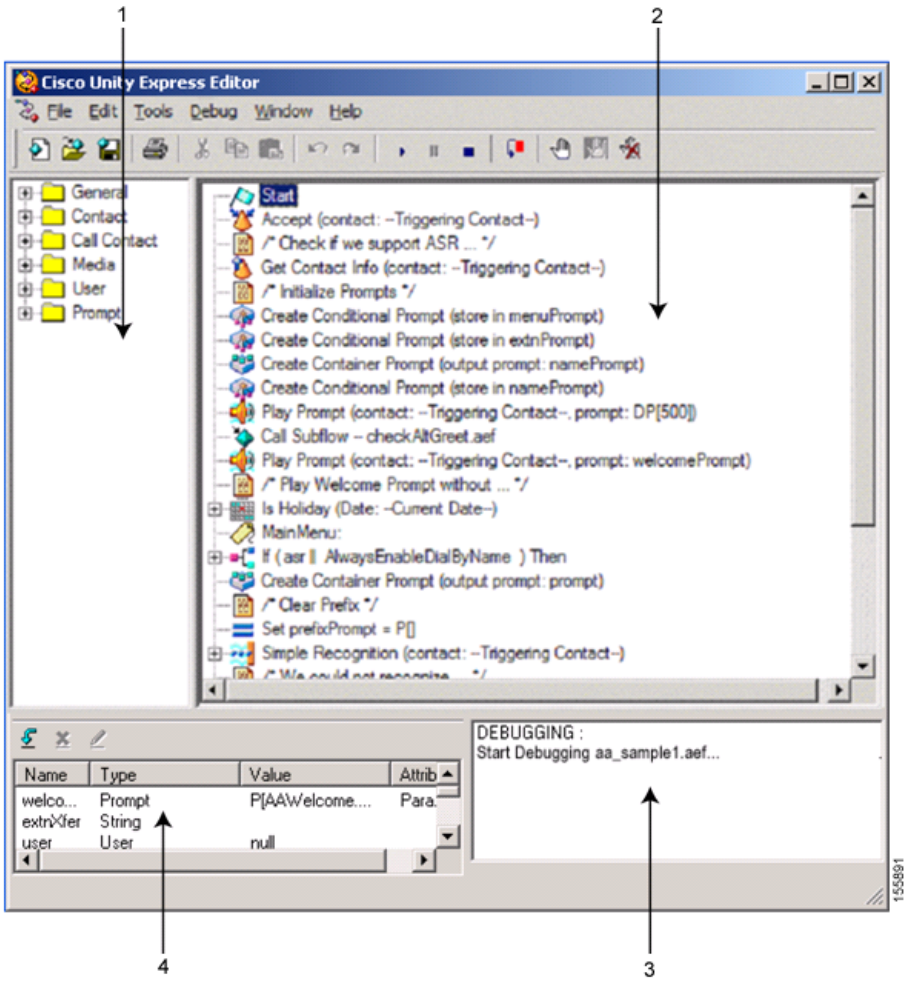
Overview of the Cisco Unity Express Script Editor

The Cisco Unity Express Script Editor is a visual programming environment for creating auto attendant application scripts.

Figure 2 shows the Cisco Unity Express Script Editor window, which is divided into four panes:

1. Palette
2. Design
3. Debug
4. Variable

Figure 2 Cisco Unity Express Script Editor Window



Palette Pane

Use the **Palette** pane to choose the steps you need for creating your script. To expand the contents of a Palette tree, click the plus sign (+) to the left of the Palette folder icon in the **Palette** pane.

Note If you try to drag a step to the **Design** pane when a customizer window is open, the **Design** pane will not accept the step. Before you drag a step to the **Design** pane, close any open customizer window(s), one or more of which may be hidden behind the Cisco Unity Express Script Editor window.

Design Pane

To create your script, click a step in the **Palette** pane and drag it on top of the step that it should follow in the **Design** pane. Each step performs a specific function and creates a portion of the underlying programming. You can customize most of the steps after you have placed them in the **Design** pane.

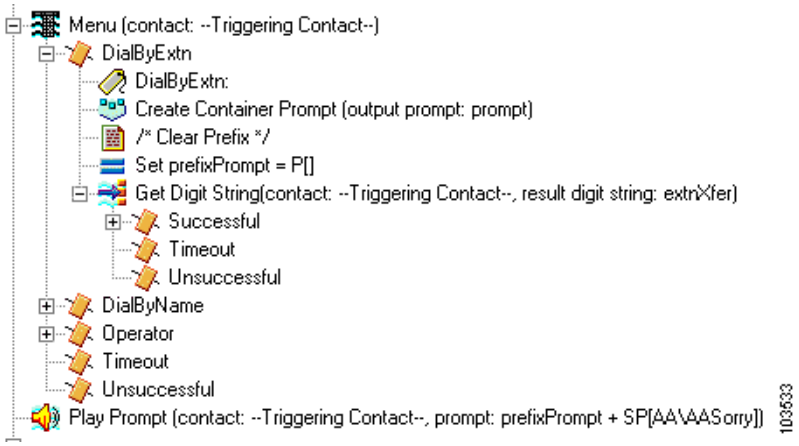
To add a step to your script, drag the step icon from the **Palette** pane and drop it onto the step that it will follow in the **Design** pane. Place the steps in logical order for the script that you are building.

To change the order of a step in the script, drag the individual step icon from its existing location to its new location. To delete a step, select the step icon and press the **Delete** key.

To end the script, click the **General** palette and drag **End** to your script. The **End** step appears.

Many steps have output branches under which you can add steps to provide the desired script logic based on the **End** condition of the step.

Figure 3 Script Example: Design Pane



As shown in the expanded **Menu** step in Figure 3, the **Menu** step has five output branches:

- DialByExtn
- DialByName
- Operator
- Timeout
- Unsuccessful

Output branches often contain steps and other output branches. The **DialByExtn** output branch in Figure 3 contains five steps below it, one of which (the **Get Digit String** step) contains three output branches.

To expand the script under a step, click the plus sign (+) to the left of the step icon. To hide the script under a step, click the minus sign (–) to the left of the step icon.

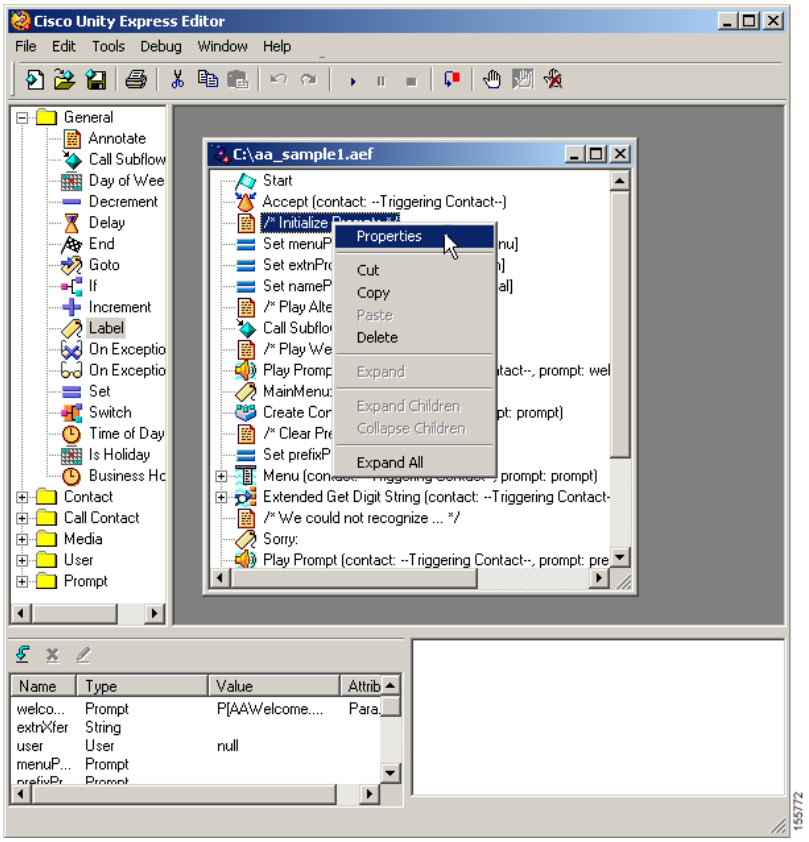
At run time, each script follows a hierarchical sequence, as indicated by the vertical lines connecting steps. In Figure 3, for example, if the script reaches the **Timeout** output branch of the **Get Digit String** step, the script moves to the next step at the level of the **Menu** step that, in this example, is the **Play Prompt** step.

Use the **Design** pane to create your script. Drag script steps from the **Palette** pane to the **Design** pane.

Step Properties

Most steps have properties that can be modified according to the needs of the script. Depending on the step, the properties can be grouped under multiple tabs. To display the properties window for a step, right-click the step in the **Design** pane and click **Properties** in the popup menu, as shown in the **Label** step **Properties** dialog box example in Figure 4.

Figure 4 Properties Popup Menu: Label Step



Variable Pane

In the **Variable** pane add and modify the script variables. In the **Variable** pane *Variables* store data that a script uses when it executes the steps. Any step in your script can use variables after you define them in the **Variable** pane of the Cisco Unity Express Script Editor window.

You can also map variables you define for your script to variables you define in a *subflow*. A subflow is a set of steps that function as part of another script, called the *primary script*. A subflow can use and manipulate a variable, and then return the data that is stored in the variable to the primary script. Scripts cannot share variables with other scripts, except in the case of default scripts, where the primary script automatically transfers the values of its variables to a default script.

The value of a variable can change during execution.

Variables

Variables store user-defined data or data resulting from the completion of a step or expression. Any step in your script can use a variable after it has been defined. Because data comes in different forms, you must also define the variable type before you can use it. Variables are grouped into the following basic built-in variable types (see the "Basic Built-in Variable Types" section on page 15):

- Boolean
- Character
- Float
- Integer
- String
- Date
- Time
- BigDecimal
- BigInteger
- Double
- Long

Defining Variables

Because data comes in different forms, you must also define the variable before you can use it. Click the:

1. **New Variable** icon at the top left corner of the **Variable** pane to define a new variable.

   The **Edit Variable** window appears. After you use the **Edit Variable** window to define your variables, the variables appear in the **Variable** pane.
2. **Delete Variable** to delete the selected variable.
3. **Modify Variable** to change the variable to the selected variable.

Figure 5 Variable Pane and Edit Variable Window

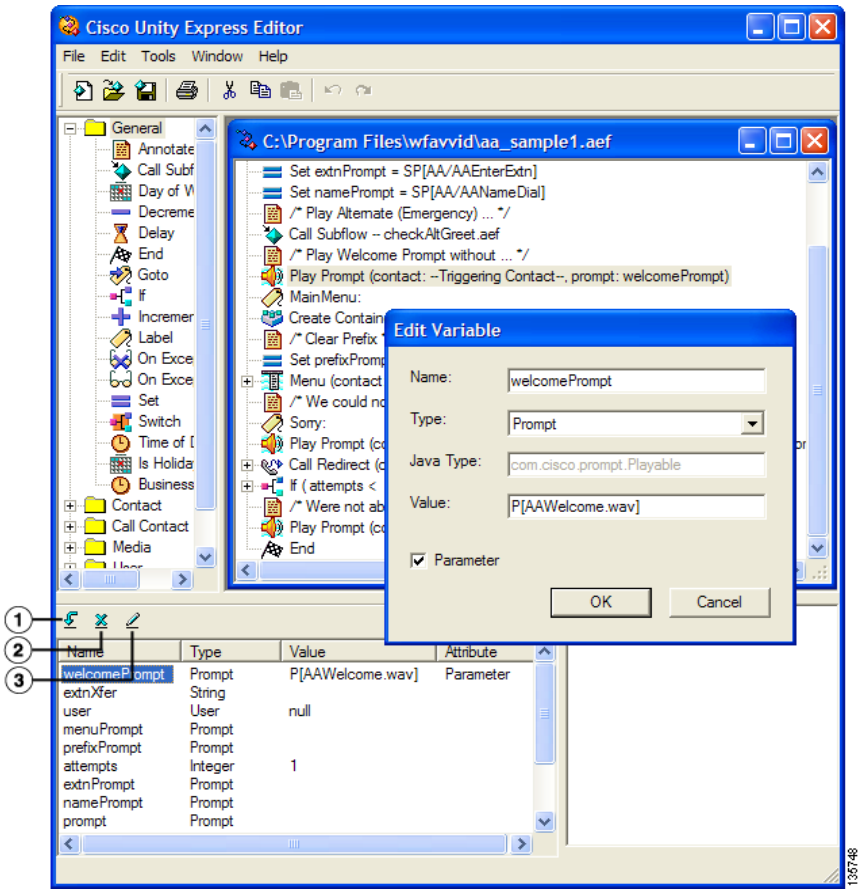


Table 3 describes the fields in the **Edit Variable** window.

Table 3 Edit Variable Properties

Property	Description
Name	Name of the variable you want to define.
Type	Type of variable that you want to declare. See the "Basic Built-in Variable Types" section on page 15 for the available variable types.
Java Type	Fully qualified class name located by using the CLASSPATH environment variable on your computer. **Note** The field displays the actual Java type for the built-in data type chosen in the **Type** drop-down menu.
Value	Data you initially assign to a variable. The type of data you enter must match the data type you declared in the **Type** field.
Parameter	If the parameter variable is checked, it sets the value for this parameter in the auto attendant web interface when you use the Cisco Unity Express Script Editor.

Basic Built-in Variable Types

You must set the type for each variable you define. A variable type indicates the kind of information that a variable contains and allows the Cisco Unity Express Script Editor to process that information accordingly. For instance, the script uses a variable containing the string “Tuesday” differently than it uses a variable containing the number 25.

Table 4 describes the basic built-in variable types.

Table 4 Variable Types

Variable Type	Description
Boolean	Variable that is either true or false, and is used primarily by the **If** step in the **General** palette of the Cisco Unity Express Script Editor: • t, f • true, false
Character	Consists of alphanumeric characters: • Lowercase letters a to z • Uppercase letters A to Z • Digits 0 to 9 • Any escape sequence: “\t”, “\r”, “\0”, “\n”, “\f”, “\\”, “\” • “\uXXXX” can be used to represent any character using the character hexadecimal Unicode number XXXX
Float	floating point variable that includes decimal numbers. All floating point values are stored as Double values. This feature prevents any loss in precision by how Java stores Float values. If a value cannot be stored as a Float, the value is automatically stored as a BigDecimal value, with some loss in precision.
Integer	Consists of whole numbers from –2147483648 to and including 2147483647. The script first parses the value as an integer. If this attempt is unsuccessful, the script parses the value as a long variable. If this attempt fails, the script parses the value as a BigInteger variable. If the script cannot represent the value as an Integer, the result may be unknown.
String	Consists of a set of Unicode characters from “\u0000” to and including “\uffff”: • “Hello”, “C:\WINNT\win.ini”. This format does not support any escape characters or Unicode characters. • u“\”This is a quoted string\””, u“\tHello”, u“\u2222\u0065”, u“C:\\WINNT\\win.ini”, and so on. This format supports the same escape sequences or Unicode characters described for the Character type.

Table 4 Variable Types (continued)

Variable Type	Description
Date	Contains date information as follows: • D[12/13/03] • D[Dec 13, 2003] • D[January 20, 2003] • D[Tuesday, April 12, 2003] • D[12/13/03] • D[12/13/03 5:50 PM] • D[April 1, 2003 12:00:00 AM PST] The parameter that you specify within the D brackets is parsed based on any combination of the following two formats: • "<date>" • "<date> <time>" The Cisco Unity Express Script Editor supports four <date> specification formats: • SHORT (12/13/03) • MEDIUM (Jan 12, 2003) • LONG (January 12, 2003) • FULL (Tuesday, April 12, 2003)
Time	Contains time information: • T[3:39 AM] • T[11:59:58 PM EST] The parameter specified inside the T brackets is parsed based on the format "<time>". The Cisco Unity Express Script Editor supports three <time> specification formats: • SHORT, such as "3:30 PM". • MEDIUM, such as "3:30:32 PM". • LONG and FULL (which are identical), such as "3:30:42 PM PST".
BigDecimal	Consists of an arbitrary-precision integer with a scale, where the scale is the number of digits to the right of the decimal point: • 3.14159 • 2E-12 • –100
BigInteger	Represents arbitrary-precision integers: • 234556789 • 0 • –23

Table 4 Variable Types (continued)

Variable Type	Description
Double	Represents an expanded Float variable. If the script cannot hold the value as a Double, the script automatically stores it as a BigDecimal: • 3.14159 • 2E–12 • –100
Long	An expanded Integer variable. The script first parses the value as a long variable. If it fails, the script parses the value as a BigInteger. If the script cannot represent the value as a long variable, the result may be unknown: • 234556789 • 0 • –23

Exporting Variables by Using Parameters

To declare variables as parameters, click **Parameter** in the Edit Variables dialog box. The set parameter feature allows you to set the value for a parameter in the auto attendant (AA) web interface. Because the value is initialized at configuration time for the script that uses it, you can change the value without editing the script in the Cisco Unity Express Script Editor. Such a variable is called an *exported variable* or *parameter.*

For example, when you add a new automated attendant by using the AA Wizard, the second window of the AA wizard (the Script Parameters window) provides a list of the parameters with their default or current values. You can modify the values in this list.

The variable types that Cisco Unity Express supports for parameters include Number, Character, String, Boolean, and Prompt.

Contact Variable

A Contact variable consists of a contact that represents a telephone call. You can pass a Contact variable as a parameter to a subflow.

Prompt Variable

A Prompt variable contains the information that the script uses to create prompts for callers. A Prompt variable can be as simple as a single prompt or as complex as a concatenation of multiple prompts.

User Variable

A User variable contains useful information for user authentication. You cannot manually enter a User variable as a value. User variables can be returned only from the **Name To User** step of the **Media** palette. You can pass a User variable as a parameter to a subflow.

Debug Pane

The Debug pane is not available in this version.

Using Prompts

The Cisco Unity Express Script Editor uses the following two types of prompts:

- System prompts: Used internally by Cisco modules and Cisco sample scripts. System prompts are used internally by the system.

  Note There is no guarantee of the continued availability of any system prompt in future releases.

- User prompts: Defined by the user and managed by the administrator using the **Voice Mail > Prompts** web page of the Cisco Unity Express GUI administrator interface or by calling in to the Greeting Management System. The script retrieves both user and system prompts from the Prompt Repository.

All **Media** and **Prompt** steps support prompts specified in the following ways:

- String expression: User-defined prompts that are located in the User Prompts directory.
- Prompt expression: Dynamically created at run time.

Note You must define all prompts played back and recorded with a RIFF header of type WAVE and G711 u-law format.

For more information on managing the prompts, see see the *Cisco Unity Express GUI Administrator Guide* or the *Cisco Unity Express Voice-Mail and Auto-Attendant CLI Administrator Guide for 3.0 and Later Versions*.

Using Expressions

Expressions are useful if you do not know the exact prompt value at design time and instead would rather enter a formula that can be evaluated later at run time. The resulting type of the expression must match the expected input type or types (which you check at design time).

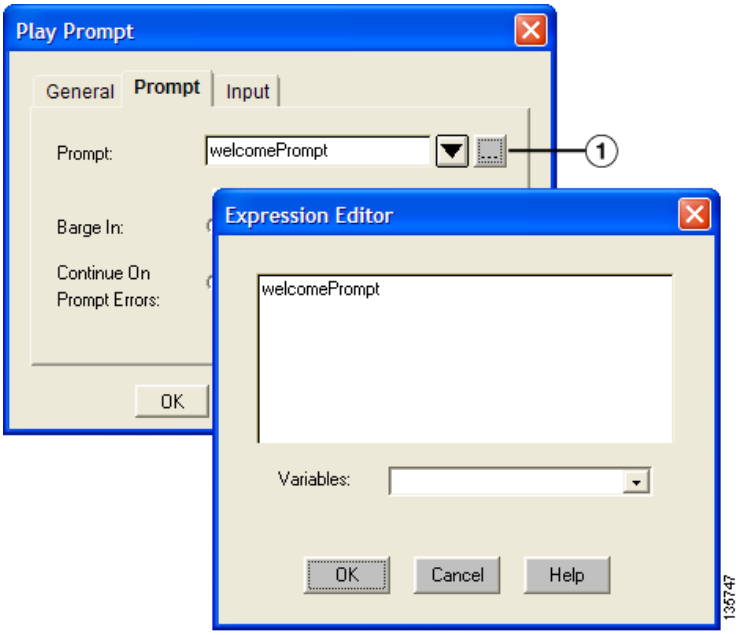
Many steps include an Expression Editor (...) button (1 in Figure 6) in the customizer window, which you can use to enter an expression.

You can enter an expression directly in the input text field, or click Expression Editor (...) to open the Expression Editor.

You can enter the expression in the text field, or choose from the Variable drop-down menu to get quick access to variables you have previously defined in the script. After you choose a variable from the **Variable** drop-down menu, the variable name appears in the input text field.

After you enter the expression, click OK. The Expression Editor closes.

Figure 6 Expression Editor Button and Expression Editor



Using Operators

To create expressions, use operators to combine variables and other values (also known as operands) to produce a result that is not known until the script is run. Operator priority refers to the order in which the operators are evaluated if there is more than one operator in an expression. The following operators are listed in order of priority:

1. Parentheses (...): Works with any expression and allows you to give priority to the expression contained in the parentheses.
2. Multiplication (*), Division (/): Works with any number expression (integer, long, float, decimal, BigInteger, BigDecimal).

   Number operands are properly promoted to a valid type before testing.
3. Addition (+), Subtraction (–): Works with any number expression (integer, long, float, decimal, BigInteger, BigDecimal).

   Number operands are properly promoted to a valid type before testing.
4. Less Than (<), Greater Than (>), Less Than or Equal (<=), Greater Than or Equal (>=)

   Comparison operands work only on String, Character, and Number operands.
5. Equal to (==), Not Equal to (!=)

   Testing for the <null> constant is supported by the two equality operators.
6. And (&&): Works only with Boolean expressions.
7. Or (||): Works only with Boolean expressions.
8. Concatenation (+).

If at least one of the operands is a String, then the other one if it is not a prompt, one is converted to a String by using the String.value() method. The result is a new String corresponding to the concatenation of the String representation of both operands. Typically, the String.valueOf() method simply calls the toString() method of the object being concatenated, or returns the string "null" if the object is null.

If the operands are Characters, then they are concatenated together, resulting in a new String.

Handling Exceptions

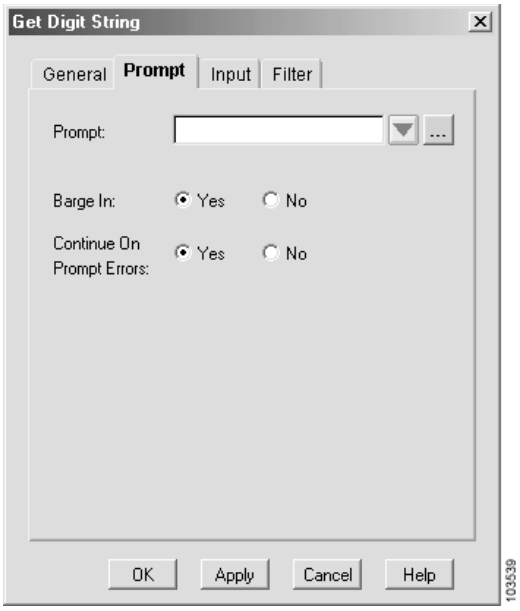
The Cisco Unity Express Script Editor provides a variety of ways to handle exceptions. Exceptions are errors in scripts from an unexpected user input or an unexpected result in scripts.

Continue on Prompt Errors Step

The **Continue on Prompt Errors** option allows the script to continue to run when the script receives invalid input (for example, Invalid Audio Format or File Not Found).

The Cisco Unity Express Script Editor provides the **Continue on Prompt Errors** option in the customizer windows of steps in the Media palette. (See the "Media Steps" section on page 125.) For example, Figure 7 shows the **Prompt** tab of the **Get Digit String** customizer window.

Figure 7 Continue on Prompt Errors Option: Prompt Tab of the Get Digit String Customizer Window



When enabled, the step continues with the next prompt in the list of prompts to be played back. If the prompt is last in the list, the script waits for caller input.

When you enable **Continue on Prompt Errors**, you instruct the script to ignore prompt errors and continue as if the playback of a particular prompt was successful. For example, in a sequence of prompts "1 + 2 + 3", if prompt #1 fails, the step will continue with prompt #2. If prompt #3 fails, the step continues to wait for caller input as if prompt #3 had been properly played back.

When you disable **Continue on Prompt Errors**, the media steps generate an exception, which can then be handled in the script.

Available prompt exceptions are the following:

- InvalidPromptArgumentException
- PromptException
- UndefinedPromptException
- UndefinedPromptGenerator
- UnsupportedPromptExpression

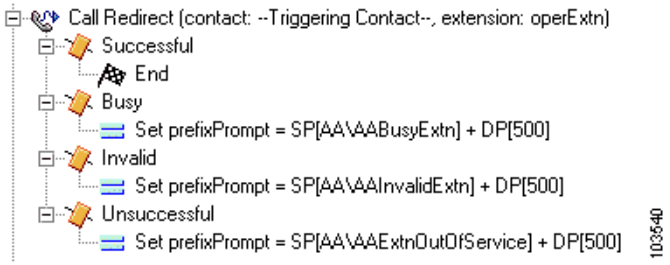
Error Output Branches

Error output branches are parts of a script that provide instructions on what to do when typical errors occur.

Figure 8 shows error output branches under a **Call Redirect** step in a script.

Note The script provides error branches only for expected error conditions, not for system errors.

Figure 8 Error Output Branches: Call Redirect Step



In this figure, the **Call Redirect** step includes logic for both an invalid extension and an out-of-service extension.

On Exception Goto Step

The **On Exception Goto** step of the **General** palette (see the "On Exception Goto" section on page 110) causes the script to continue running at a specified place in the script when an exception is generated.

By using the **On Exception Goto** step for a specific exception in a script, you can register a new handler for a specific exception or override a previously existing one.

The registration process affects the complete script. The assigned handler activates the script regardless of whether the exception occurs (before, during, or after the given step). After the step runs, the handler is registered until either a new handler is reregistered or the exception is cleared with the **On Exception Clear** step.

If an exception results in a subflow, the script first consults the exception handlers of the subflows. If none are defined for the given exception, the exception aborts the subflow, and the Cisco Unity Express application looks for exception handlers in the parent script. This process continues until the script finds an exception handler or the top level of the script is reached.

If no exception handlers are registered, the script aborts and error handling falls back to the last level of error handling, which is the default script.

Using Default Scripts

The default script is the last level of user-defined error handling before the Cisco Unity Express Script Editor applies a default system treatment to all active contacts.

The Cisco Unity Express Script Editor invokes this default script under the following conditions:

- The main script aborts, which happens for either of the following reasons:
  - An uncaught exception occurs.
  - The Cisco Unity Express application software is unable to invoke the primary script because it has not been properly validated.
- An incoming call must be aborted because the Cisco Unity Express application software has reached its limit for the number of simultaneous sessions for the application.

In each of these scenarios, the Cisco Unity Express Script Editor marks all active contacts as aborting before the default script is run. The final state of these contacts is Aborted, even if the contacts are transferred or redirected as a result of the default script running.

Caution The purpose of the default script is to gracefully terminate the call when the main script fails, not to have a fallback to provide the original services intended by the primary script. This distinction is important because using system resources to run this default script may impair system performance. If the primary script fails too often, fix the primary script instead of providing another script to attempt the same task.

The default script does not run if the primary script ends normally. If contacts are still active when the primary script ends, all active contacts not marked as handled will abort, and all active contacts marked as handled are simply terminated. In this case, check the primary script for any design problems.

Note The default script provides only a final feedback to the contact regarding the system problem and does not continue the service or restart the service.

The system applies the CallContact script if the contact is still active after the system executes the default script (if any). The CallContact script plays back the prompt, “We are currently experiencing system problems, please call back later” as an announcement, followed by a fast busy signal.

Script Interruption

Script interruption is a feature that allows external events to interrupt the current processing of a script in order to return to another part of the script or stop the execution of the script.

Use script interruption typically when the script needs to be notified that one of its contacts has been remotely terminated, such as when the caller hangs up.

Note In every case, any event that triggers the need to interrupt the script can occur at any time while the script executes other steps.

By default, scripts are automatically interruptible before any step is executed. If any external event (such as that described in the preceding text) interrupts the script, the script continues running based on the proper handling for the specific event before the script continues with the next step.

If you want two consecutive steps to run without the possibility of interruption, you must move these two steps to a subflow where you can disable interruptions completely while the script processes that subflow.

Cisco Unity Express Script Editor has an "interruptible" option for some steps that allows you to indicate whether or not the script can interrupt the step from within if an external event occurs.

When a contact terminates remotely, the script performs one of the following actions:

- When a caller hangs up, the script is interrupted (if possible) and a ContactInactiveException is generated. This exception can then be handled with the **OnExceptionGoto** step of the **General** palette.
- When a caller hangs up and no exception handling logic is available, the script immediately aborts.
- When managing multiple contacts, the **OnExceptionGoto** step cannot differentiate which contact was remotely terminated. Instead, it must specify a Label to which it can loop through all known contact variables and use the **Get Contact Info** step of the **General** palette to search for an Active flag.

If an interrupting event occurs when the script is not currently interruptible, the script is automatically interrupted whenever it becomes interruptible again. For example, although a script is not interruptible when it is running a subflow that is marked to disable interruptions, the script processes the interruption as soon as the subflow terminates, and control is returned to the parent (if that primary script is interruptible).

Installing the Cisco Unity Express Script Editor

This section describes how to install the Cisco Unity Express Script Editor application.

Note Do not install the Cisco Unity Express Script Editor application on hardware on which the Cisco Customer Response Solutions (Cisco CRS) Editor application is currently installed. These applications share registry files and will not work if installed on the same hardware.

The Cisco Unity Express Script Editor is a Microsoft Windows application. The computer you install it on must be running one of the following operating systems:

- Windows NT (Workstation or Server) with Service Pack 4 or later
- Windows 2000 (Professional or Server)
- Windows XP Professional

Download the Cisco Unity Express Script Editor installation program from Cisco.com or install it from the Cisco Unity Express Application Software CD. The filename is Cisco Unity Express Editorx.x.x.exe, where x.x.x is the version that you are installing.

Follow these steps to install the Cisco Unity Express Script Editor:

Step 1 Double-click the installation program file.

The InstallShield Wizard appears and begins extracting files for the installation. (This process may take a few minutes.)

Step 2 Follow the prompts to install the application. A default installation is acceptable for most users: click Yes and Next buttons when prompted.

The prompts also allow you to move back to an earlier step in the installation process or cancel the installation completely.

Step 3 To verify that the application is installed correctly, start the application: on the task bar click **Start**, All Programs, Cisco Unity Express Developer, Cisco Unity Express Script Editor.

The default installation path on your hard drive is `C:\Program Files\wfavvid\WFCCNEditor.exe`.