Tip: MSI Properties and Deferred Execution

Robert Dickau
Senior Technical Trainer

Abstract

This article describes how to get the value of an MSI property from within a deferred custom action.

Background

A Basic MSI installation program does not use an explicit script to drive the installation, but instead uses sequences of actions to determine the dialog boxes and operations the installation program should display and perform. In this sense, MSI actions are analogous to function calls in a typical programming language.

There are two sequences used by a typical installation program: the User Interface sequence and the Execute sequence. The User Interface sequence displays dialog boxes and queries the target system, but does not make any system changes. The Execute sequence performs system changes, but does not display a user interface.

Analogous to variables in a programming language are Windows Installer properties. Windows Installer defines dozens of predefined properties, and an installation author can define custom properties to store any extra data needed at run time. A property can, for example, be set by the user in a dialog box, and then later be written to the registry.

Property names are case sensitive. Two classes of MSI properties are public properties, which have names containing only uppercase letters (examples are USERNAME and INSTALLDIR); and private properties, whose names contain at least one lowercase letter (as in AdminUser and ProgramFilesFolder). The values of public properties can be set at the command line, and their values are preserved when execution switches from the User Interface sequence to the Execute sequence. Private properties, on the other hand, cannot have their values set at the command line, and are reset to their default values when execution switches from the User Interface sequence to the Execute sequence.

During installation, the Execute sequence runs in two stages, immediate mode and deferred mode. Immediate mode walks through the actions in the Execute sequence, generating an internal, hidden installation script; this script contains, for example, instructions for what files, registry data, shortcuts, and so forth, to install. Immediate mode does not touch the target system. Note that the User Interface sequence runs only in immediate mode.

The second stage of the Execute sequence, deferred mode, carries out the system changes described by this internal installation script. (Strictly speaking, deferred mode is performed only for actions between the built-in actions InstallInitialize and InstallFinalize.) During deferred execution, MSI property values are fixed and cannot be changed. Moreover, the values of only a handful of MSI properties can explicitly be read during deferred execution.

Getting Property Values in Custom Actions

When you need to extend the behavior of an installation program, you can write one or more custom actions. MSI supports custom actions that launch executables, set MSI property values, and call various

types of DLL and script functions.

During immediate execution, a VBScript custom action can read the value of an MSI property using the Property property of the Session object. For example:

```
' get a property value during immediate mode
MsgBox "Right now, USERNAME is: " & Session.Property("USERNAME")
```

Similarly, a C or C++ DLL or an InstallScript custom action can call the MsiGetProperty API function to read the value of a property.

As mentioned above, however, getting the value of an MSI property during deferred execution is somewhat more difficult, as deferred actions have access to only a very few built-in properties: ProductCode, UserSID, and CustomActionData. (Note that this statement is untrue for built-in types of actions that use property values as arguments. For example, a deferred launch-an-EXE action that uses "[INSTALLDIR]Readme.txt" as its argument will have the INSTALLDIR property resolved during immediate mode and fixed during deferred mode. It is only custom actions that explicitly read a property value using Session.Property or MsiGetProperty that need to use the technique described here.)

It is this last property, CustomActionData, that is used to read a property value in a script during deferred mode. For an example, suppose you have a deferred VBScript custom action called "ReadPropDeferred", in which you want to read the value of the USERNAME property. The steps involved in populating this property are the following:

1. Create an immediate-mode set-a-property custom action that sets a property called ReadPropDeferred to the value of the USERNAME property. That is, in the Custom Actions view, create a set-a-property action with property name **ReadPropDeferred** and value **[USERNAME]**, and schedule the action somewhere before the ReadPropDeferred action. The main idea is that the *property* name here is the same as your deferred *action* name.

2. In the deferred VBScript code of the ReadPropDeferred action, use Session.Property ("CustomActionData") to read the desired value:

   ```
   ' get property value during deferred mode
   MsgBox "Right now, USERNAME is: " & Session.Property("CustomActionData")
   ```

At run time, the immediate action that sets the ReadPropDeferred property populates CustomActionData for that specific deferred action; and the deferred ReadPropDeferred action reads CustomActionData (instead of USERNAME) to determine the desired data. Of course, this same technique can be used with DLL custom actions that use MsiGetProperty to read property values.

As mentioned above, during deferred execution the installation script is fixed, and therefore property values cannot be set from within a deferred custom action.