Destination and Session Management for the SAP DCOM Connector

Thomas G. Schuessler

Thomas G. Schuessler is the founder of ARAsoft, a company offering products, consulting, custom development, and training around the world. The company specializes in integration between SAP and non-SAP components and applications. Thomas is the author of SAP's CA925 "Programming with BAPIs in Visual Basic" class. Prior to founding ARAsoft in 1993, he worked with SAP AG and SAP America for seven years.

(complete bio appears on page 50)

The SAP DCOM Connector (SDC) is the standard middleware for Microsoft Windows[1] applications that require access to SAP functionality through BAPIs or other functions.[2] These applications could, for example, be desktop programs or Internet applications that provide an alternative frontend to any transaction in the SAP system.

In order to get at the functionality in an SAP system, a client program needs to log on, identifying a particular SAP system and providing valid values for the client, userid, password, and optionally, the language to be used. SDC allows you to store information about an SAP system and a user on the client system, using the notion of a *destination*. A destination identifies an SAP system and optionally contains default values for the client, userid, password, and language. Destinations are stored in the Windows Registry. SDC contains an API — the CCRegistry API — to access these destinations from a client program. On top of that API, SDC provides a destination management screen (accessible through Internet Explorer) as part of its administration facilities.

Based on these SAP-supplied capabilities, several opportunities arise to build reusable components and applications:

- Building a component that encapsulates the CCRegistry API in a more object-oriented fashion would make it easier for developers to

[1] Windows 95, 98, NT, 2000.

[2] The protocol for communication between SAP and non-SAP systems is called RFC (Remote Function Call). BAPIs (Business Application Programming Interfaces) are RFC-enabled function modules (RFMs) defined as methods in the Business Object Repository (BOR). In addition, there are other RFMs that are not BAPIs but can still be used from external applications.

work with destinations in their programs and components.

- Some applications log on to R/3 transparently (without any user interaction). Many, if not most, applications, though, will need to present users with a list of destinations to choose from and provide an opportunity to view and change the default user information (defined in the destination) to be submitted to R/3. The current version of SDC (4.6B) does not supply a component for this task. Building a reusable component, encapsulating the Registry access and the user interaction, would prevent application developers from having to spend time writing their own component or — even worse — writing slightly different logon code for each SDC-based application.
- In some companies, maintenance of the destinations on client systems will be entrusted to administrators only. Other companies may want their users to be able to define and change their own destinations. Using standard SDC, this requires the installation of Internet Explorer and the complete SDC administration facilities on the user's machine. Building an application that only allows destination management would solve the dilemma of either giving the user all or no administration capabilities.

There is one related subject that I want to cover in this article. Once a client program has logged on to R/3, it is often desirable that it has access to session-specific information — for example, the language code selected by the user during logon, or whether or not the user is still connected to R/3. So I will also show you how to retrieve information about an active session with R/3 at runtime.

This article will show you how to accomplish all four tasks, with coverage of:

- Destinations in SDC — an overview
- The CCRegistry API
- Building a component to encapsulate the CCRegistry API
- Building a destination management application
- Building a visual logon component
- Getting information about a session at runtime

The programming language used for the sample code is Visual Basic, but if you are working with another Windows-enabled language, you should have no problem making use of the concepts that are presented here.[3]

Freeware and Buyware

This article provides more than just a theoretical treatment of these tasks. I have actually written a lot of software based on the ideas presented here. Some of it is free. You can download the ARAsoft DCOM Connector Destination Manager and the ARAsoft DCOM Connector Logon Component from SAP's Web site (see the appropriate sections of the article for the URLs). Both pieces of software are freeware. As for the encapsulation of the CCRegistry API and the session information retrieval, I have supplied generous amounts of source code in this article. You can use it as the basis for your own solutions, or license my ARAsoft DCOM BAPI Object Factory, which contains many additional capabilities related to SDC-based BAPI programming. To obtain a free trial copy, just send me an e-mail.

Destinations in SDC — An Overview

A destination in SDC is used to identify an R/3 system and, optionally, define default user information. (Multiple destinations can refer to the same SAP system.)

[3] In order to benefit from this article, you should already know the basics of DCOM Connector programming. An overview of these basics was provided in my article "Programming with BAPIs using the SAP DCOM Connector," which was published in the January/February 2000 issue of the *SAP Professional Journal*.

Figure 1 Selecting an Individual Application Server

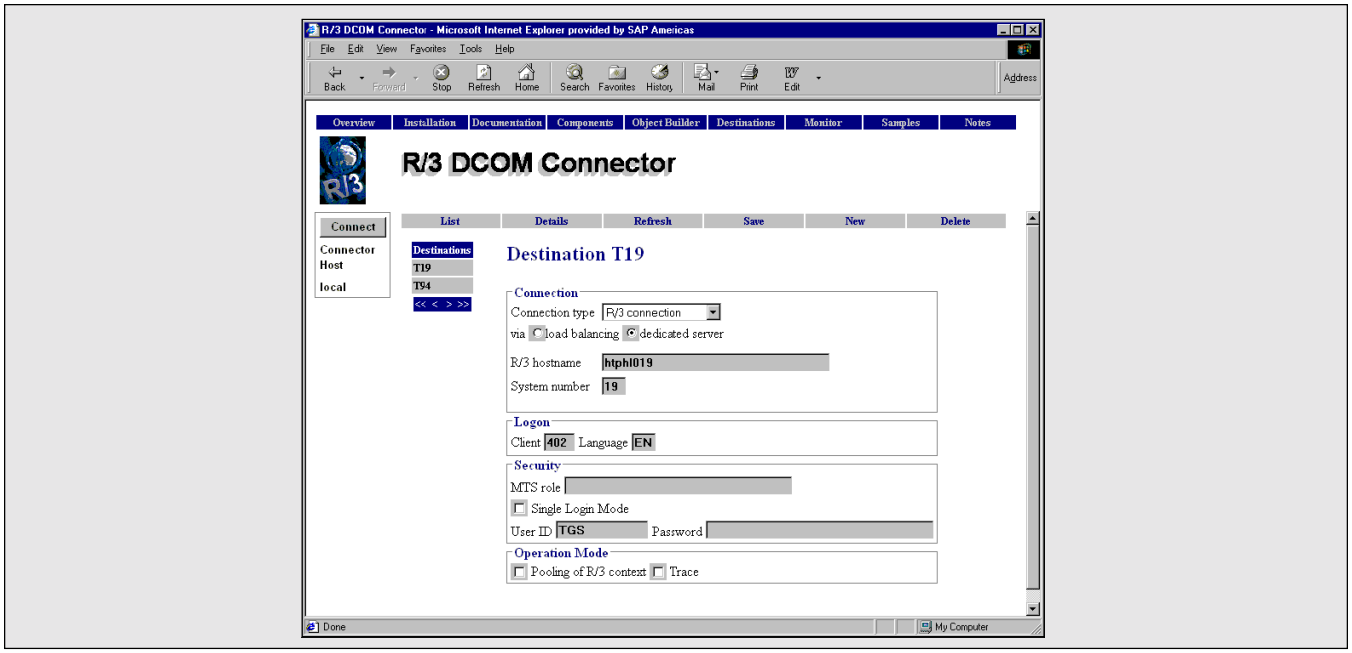
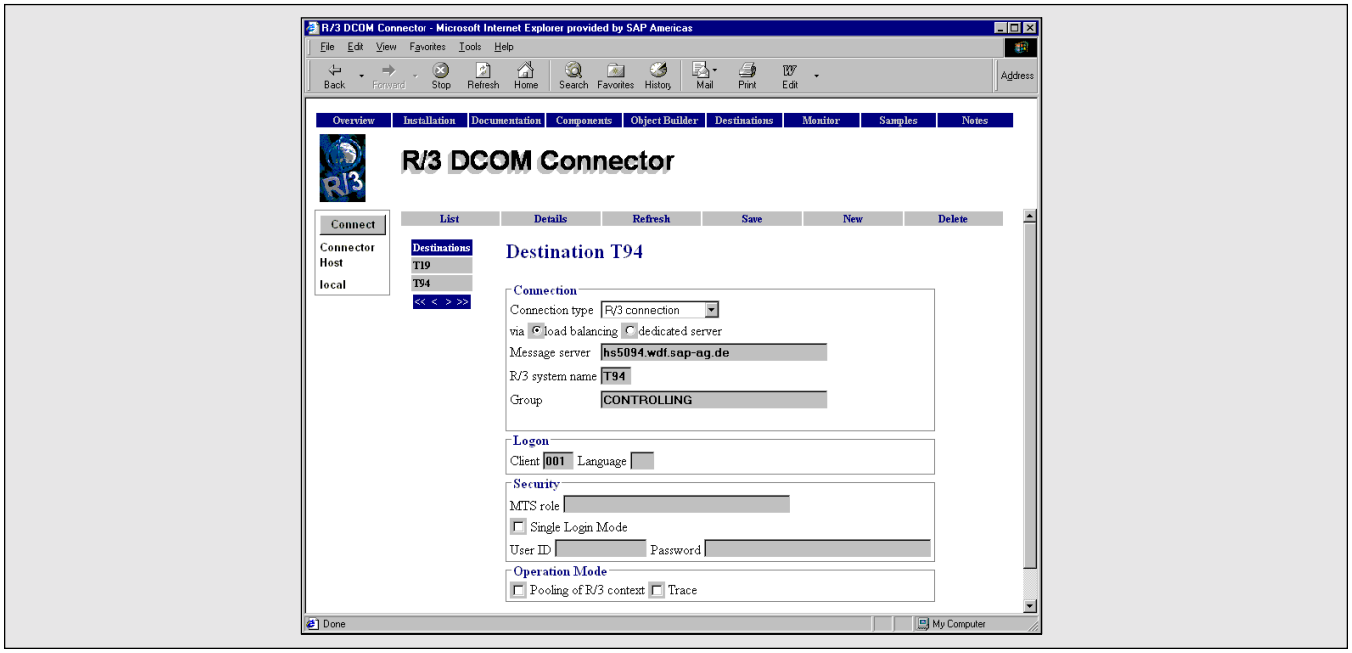


Figure 2 Selecting a Load Balancing Group

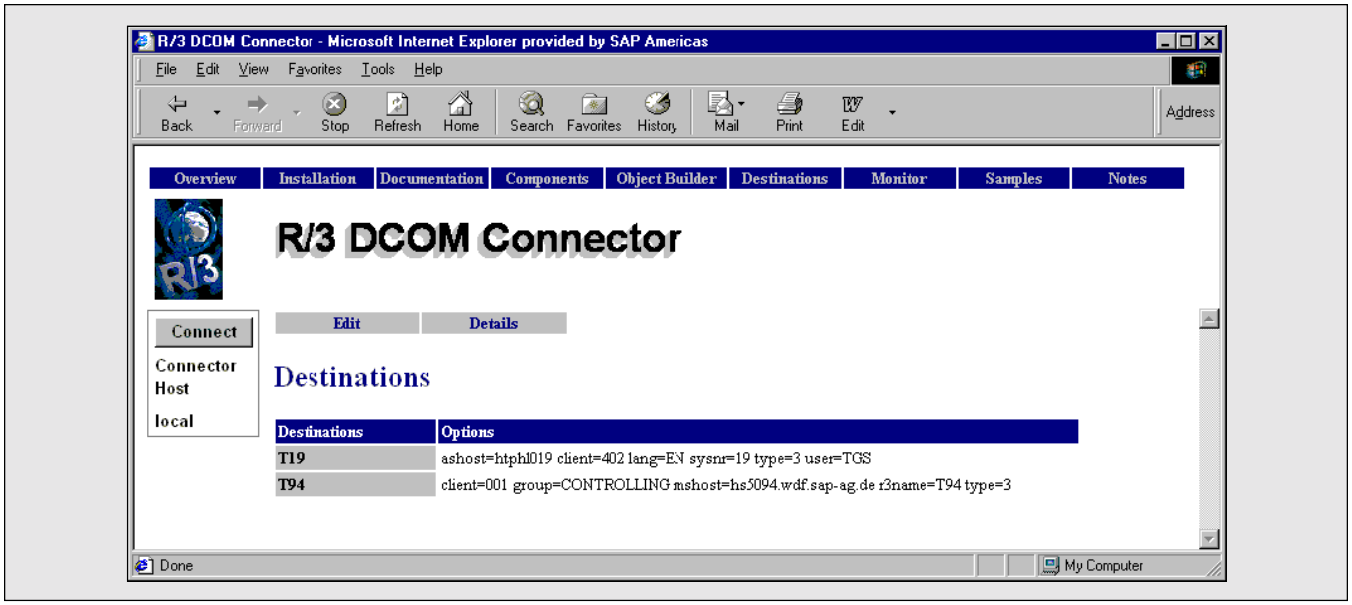


An R/3 system can be identified by either:

- Referring to an individual application server, specifying its host name and system number, as shown in **Figure 1**.
- Referring to a load balancing group,[4] specifying the message server, system name, and group name, as shown in **Figure 2** (note that this

[4] Load balancing groups are typically used in larger SAP installations in order to improve performance and/or support different code pages.

Figure 3 Destination Management



approach is only applicable if the SAP system administrator has actually defined load balancing groups for a system).

Your system administrator will be able to tell you which approach you should use and give you the required information (server name, etc.).

In addition to this identification of the SAP system, you can set defaults for the following fields that are used during logon:

- Client
- Language
- Userid
- Password

Storing a default for the password is something I would advise against, unless you are talking about a test system in which everybody is allowed everything. Although the password is encrypted before it is stored, anybody with access to your PC could now log on to this R/3 system without having to specify a password.

The defaults for Client, Language, and Userid can be overwritten at runtime, so storing a default value for any of these parameters does not prevent an application or a user from using other values during logon.

If you do not specify a language during logon, the default language defined by the SAP system administrator will be used.

Figure 3 shows an overview list of the two destinations defined in Figures 1 and 2.

The CCRegistry API

The CCRegistry class is part of the SAP DCOM Connector Administration Component, which has a file name of `ccadmin.dll`. The name of its Type Library is `CCADMINLib`. This API encapsulates access to the destinations in the Registry. Any application that needs to deal with destinations must use this API either directly or through some encapsulation. SDC's Destination Management screen itself uses the API.

Figure 4 lists the important methods of the CCRegistry class that are required to maintain destinations.

Figure 4 Important Methods of the CCRegistry Class

Function GetDestinations() As Object
Function GetOptionsAsRecord(destination As String) As Object
Sub DeleteDestination(destination As String)
Sub PutNewDestination(destination As String)
Sub PutOptionsAsRecord(destination As String, pIn As Object, passwd As String)

Figure 5 The Fields in the GetDestinations Recordset

Field Name	Description
DESTINATION	Destination name
OPTIONS	Option string

Figure 6 Important Fields in the GetOptionsAsRecord and PutOptionsAsRecord Recordsets

Field Name	Description
Loadbal	Uses load balancing (0 = No, 1 = Yes)
Ashost	Application server name
Sysnr	System number
Mshost	Message server name
Group	Load balancing group
r3name	R/3 system name
Type	System type (2 = R/2, 3 = R/3)
Client	Client number
Lang	Language code
User	Userid
Trace	Trace setting (0 = Trace off, 1= Trace on)

We will use this API to:

- Show users a list of defined destinations and their attributes
- Maintain the destinations defined on a client computer

The first method you see listed in Figure 4, **GetDestinations**, returns a Recordset with one record for each destination defined in the Registry. The structure of the Recordset is shown in **Figure 5**.

The DESTINATION field contains the name assigned to the destination when it was created. The OPTIONS field contains the string shown in Figure 3 in the Options column of the list. Instead of parsing this string and extracting its various fields, we can use the GetOptionsAsRecord method.

The **GetOptionsAsRecord** method requires you to pass the name of a destination (as obtained, for example, from a call to GetDestinations) and returns — if the specified destination exists — a Recordset containing one record with several fields. The fields that are relevant for our purposes are shown in **Figure 6**.

Field "Loadbal" tells you whether load balancing is used for this destination. If load balancing is not used (as indicated by a value of 0), the fields "Ashost" and "Sysnr" identify a specific application server of an R/3 system. If load balancing is used (as indicated by a value of 1), then fields "Mshost", "Group", and "r3name" identify a load balancing group of an R/3 system.

Field "Type" identifies the system type. You can actually use SDC to call functions in an R/2 mainframe system!

Fields "Client", "Lang", and "User" contain the defaults defined for client, language, and userid, respectively. All of them are empty if no defaults were set.

Field "Trace" informs you whether or not the trace option has been enabled for this destination. Used together, methods GetDestinations and GetOptionsAsRecord allow you to first retrieve a list of all destinations and then all the attributes for each individual destination.

The **DeleteDestination** method allows you to delete a destination — identified by its name — from the Windows Registry. Obviously, this method should be used with great care!

The **PutNewDestination** method allows you to create a new destination with the specified name. To set the values for the various attributes of the new destination, use the PutOptionsAsRecord method.

The **PutOptionsAsRecord** method allows you to change the attributes of a destination. **Figure 7** lists the method's parameters. PutOptionsAsRecord can be used to change the attributes of any existing destination, including one that was only just created by PutNewDestination. Note that "passwd" is not part of the Recordset used by both the GetOptionsAsRecord and PutOptionsAsRecord methods. This is a good design decision because it prevents a program from retrieving passwords from the Registry.[5] The "passwd" parameter of the PutOptionsAsRecord method enables you to *store* a password, though.

[5] Remember that I do not recommend storing any passwords in the Registry in the first place.

Building a Component to Encapsulate the CCRegistry API

The CCRegistry API provides the functionality you need to create an application that enables administrators to add, change, and delete destinations. The problem is that this API is not very object-oriented. The attributes of a destination are not exposed as properties of a Destination class, for example. Instead, you have to deal with the Recordset shown in Figure 6 and hard-code its field names in your client application.

Since I am a firm believer in the encapsulation of anything complex into a simple-to-use component, I have designed and built a set of classes that makes it easier for a client program to interact with destinations. I am using three classes, **Destination**, **Destinations**, and **ObjectFactory** to accomplish this. As you can easily deduce from their names, the Destinations class is a collection of items of type Destination. The role of ObjectFactory will be explained later.

The Destination Class

Figure 8 shows the properties supported by the Destination class.

If you compare Figure 6 and Figure 8, you will notice a few differences (besides slightly different names in some cases):

- The UsesLoadBalancing and Trace properties of the Destination class are defined as Boolean. This is a little easier to interpret for a client program.
- The "Type" field in Figure 6 is not represented by any property. I decided that I did not want

Figure 7 The Parameters of the PutOptionsAsRecord Method

Parameter Name	Description
destination	Destination name
pIn	Recordset as defined in Figure 6
passwd	Password

Figure 8 The Properties of the Destination Class

Property Name	Data Type	Read/Write
Name	String	Yes
UsesLoadBalancing	Boolean	Yes
HostName	String	Yes
SystemNumber	String	Yes
SystemId	String	Yes
GroupName	String	Yes
Client	String	Yes
UserId	String	Yes
Language	String	Yes
Trace	Boolean	Yes

Figure 9 The Parameters of the CreateConnectionString Method

Parameter Name	Optional	Description
pPassword	No	Password (required for logon)
pUserId	Yes	Userid (String)
pClient	Yes	Client number (String)
pLanguage	Yes	Language code (String)
pTrace	Yes	Trace (Boolean)

to support R/2 in my component. If you would like to have R/2 support in your component, you will have to add suitable properties and code.

- Fields “Ashost” and “Mshost” in Figure 6 are combined into the one property, HostName, in the Destination class. This is okay since a destination either uses load balancing, or it does not. Therefore, only one host name needs to be stored.

The Destination class has one method, with the following signature:

```
Public Function CreateConnectionString(
                pPassword As String,
                Optional pUserId,
                Optional pClient,
                Optional pLanguage,
                Optional pTrace) As String
```

Figure 9 describes the method’s parameters. The *raison d’être* for this method is that the PutSessionInfo method offered by SDC’s Session class does not have a Trace[6] parameter:

[6] The RFC trace is useful when your SDC-based application is not working as expected. The generated trace file helps a developer to debug an application. All BAPI and non-BAPI function calls together with their parameters are listed in the trace file.

Figure 10 The Properties of the Destinations Class

Property Name	Data Type	Read/Write
Count	Long	Read-only
Item(vntIndexKey As Variant)	Destination	Read-only

Figure 11 Public Methods of the Destinations Class

Public Sub Add(ByVal Item As Destination)
Public Function Exists(Key As String) As Boolean
Public Sub Refresh()
Public Sub Remove(Name As String)
Public Sub Update(ByVal Item As Destination)

```
Sub PutSessionInfo(Optional destination As String,
                   Optional userid As String,
                   Optional Password As String,
                   Optional language As String,
                   Optional client As String)
```

While PutSessionInfo allows us to override the defaults stored in the destination for the userid, password, language, and client, we cannot override the trace setting stored for the destination at runtime. Fortunately, the method allows us to pass a string, with all required keywords set appropriately, instead of the name of the destination. This string needs to be surrounded by curly braces and could look like this:

```
"{ ashost=pswdf071.wdf.sap-ag.de
client=002 lang=EN sysnr=75 type=3
user=c3026902 passwd=*** trace=1 }"
```

The CreateConnectionString method of the Destination class facilitates the construction of the required string and allows us to optionally override all fields, including Trace. This now enables us to turn on the trace for a particular program execution, without having to turn it on for the destination as such (i.e., all programs using this destination) in SDC.

The Destinations Class

The Destinations class is a collection of Destination objects. **Figure 10** shows the properties supported by the Destinations class.

These are standard properties for a collection class.

Figure 11 contains the public methods of the Destinations class.

The **Add** method adds a new Destination object to the collection and also writes it to the Registry. It will fail if a destination with the name of the new destination already exists.

The **Exists** method checks whether a destination with the specified name exists.

The **Refresh** method re-initializes the collection. The only time a client program would want to call Refresh is when the destinations in the Registry were updated by some other program after the client program created the Destinations collection object.

The **Remove** method removes the destination with the specified name from the collection and the Registry. It will fail if no destination with this name exists.

The **Update** method updates the destination identified by the Item parameter's Name property with the properties of the Destination object passed in parameter Item. This is done both in the collection and in the Registry. The method will fail if no destination with this name exists.

The ObjectFactory Class

The Destination and Destinations classes are very useful once an object of type Destinations has been created. How does a client program instantiate the Destinations class? I decided not to allow the client program to create multiple instances of the Destinations class (using Visual Basic's "New" keyword). This would have led to some difficulties, because each instance would have believed that it has exclusive control over the destinations in the Registry. Instead, I have applied the design pattern known as "Singleton": All requests for an object of type Destinations always return the same instance.

This is accomplished via a method called GetDestinations. This method is offered by a class called ObjectFactory, which is responsible for the creation of objects for which we need to closely control object creation.

The GetDestinations method returns a reference to the Destinations object created by the ObjectFactory.

Summary of the Destinations Component

If you compare the native destination management API offered by the CCRegistry class with the component containing the Destination, Destinations, and ObjectFactory classes, and if you like the latter approach better, **Listing 1**, **Listing 2**, and **Listing 3** offer you an opportunity to review the source code for the Destination, Destinations, and ObjectFactory classes. If you prefer the standard API, just continue using the CCRegistry class.

(text continues on page 42)

Listing 1: The Destination Class

```
Const CLASSNAME = "Destination"
Const ERRSOURCE = LIBNAME + "." + CLASSNAME

Private sName As String

Private sUserId As String
Private sClient As String
Private sLanguage As String

Private bUsesLoadBalancing As Boolean
Private bTrace As Boolean
Private sHostName As String

Private sSystemNumber As String

Private sSystemId As String
Private sGroupName As String

Public Function CreateConnectionString(pPassword As String, _
                                       Optional pUserId, _
                                       Optional pClient, _
                                       Optional pLanguage, _
                                       Optional pTrace) As String
```

```
Dim s As String
  s = "{ type=3"
  If bUsesLoadBalancing Then
    s = s & " mshost=" & sHostName & " group=" & sGroupName & _
        " r3name=" & sSystemId
  Else
    s = s & " ashost=" & sHostName & " sysnr=" & sSystemNumber
  End If
  s = s & " passwd=" & pPassword
  If IsMissing(pUserId) Then
    s = s & " user=" & sUserId
  Else
    s = s & " user=" & pUserId
  End If
  If IsMissing(pClient) Then
    s = s & " client=" & sClient
  Else
    s = s & " client=" & pClient
  End If
  If IsMissing(pLanguage) Then
    s = s & " lang=" & sLanguage
  Else
    s = s & " lang=" & pLanguage
  End If
  If IsMissing(pTrace) Then
    s = s & " trace=" & IIf(bTrace, "1", "0")
  Else
    s = s & " trace=" & IIf(pTrace, "1", "0")
  End If
  s = s & " }"
  CreateConnectionString = s
End Function

Public Property Get Name() As String
  Name = sName
End Property
Public Property Let Name(ByVal sNewValue As String)
  If sName = "" Then
    sName = sNewValue
  Else
    Err.Raise ERR3010, ERRSOURCE, ERR3010Text
  End If
End Property

Public Property Get UserId() As String
  UserId = sUserId
End Property
Public Property Let UserId(ByVal sNewValue As String)
  sUserId = sNewValue
End Property
```



```
Public Property Get Client() As String
  Client = sClient
End Property
Public Property Let Client(ByVal sNewValue As String)
  sClient = sNewValue
End Property

Public Property Get Language() As String
  Language = sLanguage
End Property
Public Property Let Language(ByVal sNewValue As String)
  sLanguage = sNewValue
End Property

Public Property Get UsesLoadBalancing() As Boolean
  UsesLoadBalancing = bUsesLoadBalancing
End Property
Public Property Let UsesLoadBalancing(ByVal NewValue As Boolean)
  bUsesLoadBalancing = NewValue
End Property

Public Property Get Trace() As Boolean
  Trace = bTrace
End Property
Public Property Let Trace(ByVal NewValue As Boolean)
  bTrace = NewValue
End Property

Public Property Get HostName() As String
  HostName = sHostName
End Property
Public Property Let HostName(ByVal sNewValue As String)
  sHostName = sNewValue
End Property

Public Property Get SystemNumber() As String
  SystemNumber = sSystemNumber
End Property
Public Property Let SystemNumber(ByVal sNewValue As String)
  sSystemNumber = sNewValue
End Property

Public Property Get SystemId() As String
  SystemId = sSystemId
End Property
Public Property Let SystemId(ByVal sNewValue As String)
  sSystemId = sNewValue
End Property
```

```
Public Property Get GroupName() As String
  GroupName = sGroupName
End Property
Public Property Let GroupName(ByVal sNewValue As String)
  sGroupName = sNewValue
End Property
```

Listing 2: The Destinations Class

```
Const CLASSNAME = "Destinations"
Const ERRSOURCE = LIBNAME + "." + CLASSNAME

Private colX As Collection

Public Property Get Count() As Long
  Count = colX.Count
End Property

Public Function Exists(Key As String) As Boolean
  On Error Resume Next
  colX.Item Key
  Exists = IIf(Err.Number = 0, True, False)
  Err.Clear
End Function

Public Property Get Item(vntIndexKey As Variant) As Destination
  Set Item = colX(vntIndexKey)
End Property

Public Property Get NewEnum() As IUnknown
  Set NewEnum = colX.[_NewEnum]
End Property

Public Sub Remove(Name As String)
  If Exists(Name) Then
    oCCRegistry.DeleteDestination Name
    colX.Remove Name
  Else
    Err.Raise ERR3012, ERRSOURCE, ERR3012Text
  End If
End Sub

Private Sub Class_Initialize()
  Refresh
End Sub
Private Sub Class_Terminate()
  Set colX = Nothing
End Sub
```



```
Public Sub Add(ByVal Item As Destination)
  If Exists(Item.Name) Then
    Err.Raise ERR3011, ERRSOURCE, ERR3011Text
  Else
    oCCRegistry.PutNewDestination Item.Name
    AddInternal Item
    Update Item
  End If
End Sub
Friend Sub AddInternal(ByVal Item As Destination)
  colX.Add Item, Item.Name
End Sub

Public Sub Refresh()
Dim rsX As Recordset
Dim destinationX As Destination
  Set colX = New Collection
  Set rsX = oCCRegistry.GetDestinations
  rsX.MoveFirst
  Do Until rsX.EOF
    Set destinationX = New Destination
    destinationX.Name = rsX("Destination")
    destinationX.ConnectString = rsX("Options")
    AddInternal destinationX
    rsX.MoveNext
  Loop
  For Each destinationX In colX
    Set rsX = oCCRegistry.GetOptionsAsRecord(destinationX.Name)
    rsX.MoveFirst
    destinationX.UsesLoadBalancing = _
      IIf(rsX("loadbal") = 0, False, True)
    destinationX.Trace = IIf(rsX("trace") = 1, True, False)
    destinationX.Client = rsX("client")
    destinationX.GroupName = rsX("group")
    destinationX.HostName = _
      IIf(destinationX.UsesLoadBalancing, rsX("mshost"), rsX("ashost"))
    destinationX.Language = rsX("lang")
    destinationX.SystemId = rsX("r3name")
    destinationX.SystemNumber = rsX("sysnr")
    destinationX.UserId = rsX("user")
  Next destinationX
End Sub

Public Sub Update(ByVal Item As Destination)
Dim rsX As Recordset
  If Not Exists(Item.Name) Then
    Err.Raise ERR3012, ERRSOURCE, ERR3012Text
  Else
    Set rsX = oCCRegistry.GetOptionsAsRecord(Item.Name)
    rsX.MoveFirst
    rsX("client") = Item.Client
```

```
    rsX("lang") = Item.Language
    rsX("user") = Item.UserId
    rsX("loadbal") = IIf(Item.UsesLoadBalancing, 1, 0)
    rsX("trace") = IIf(Item.Trace, 1, 0)
    rsX("group") = Item.GroupName
    rsX("r3name") = Item.SystemId
    rsX("sysnr") = Item.SystemNumber
    If Item.UsesLoadBalancing Then
      rsX("mshost") = Item.HostName
      rsX("ashost") = ""
    Else
      rsX("ashost") = Item.HostName
      rsX("mshost") = ""
    End If
    rsX.Update
    Call oCCRegistry.PutOptionsAsRecord(Item.Name, rsX, "")
  End If
End Sub
```

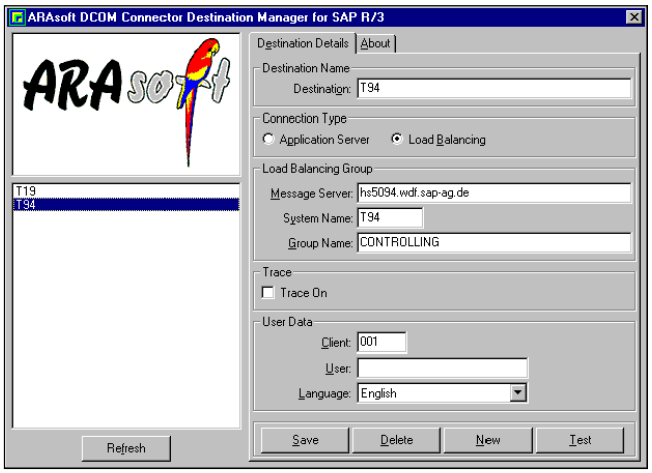
Listing 3: The GetDestinations Method in the ObjectFactory Class

```
Public Function GetDestinations() As Destinations
  If oDestinations Is Nothing Then
    Set oDestinations = New Destinations
  End If
  Set GetDestinations = oDestinations
End Function
```

Building a Destination Management Application

If normal users of SDC-based applications want to maintain their own destinations (e.g., to define a new R/3 system they want to use), they can only do so if they have access to SDC's administration features. Some system administrators might not like the idea of giving normal users complete SDC administration capabilities. On the other hand, letting users maintain just destinations will be acceptable in most cases. Since we now have a nice component (in my opinion) that encapsulates access to the destinations, it is not hard to develop an application that allows a user to do this. **Figure 12** contains a screenshot of the application I built for this purpose. This program is freeware and can be downloaded from:

Figure 12 The Destination Manager for SAP R/3



http://www.sap.com/solutions/technology/bapis/com/ara_dest.zip

Using this program, you can add, change, and delete destinations. In addition, you can test any destination (logging on to the R/3 system) to make sure that its definition is correct. The only hard thing about writing this program was making sure that the GUI was intuitive to use. The Destination and Destinations classes took care of everything else.

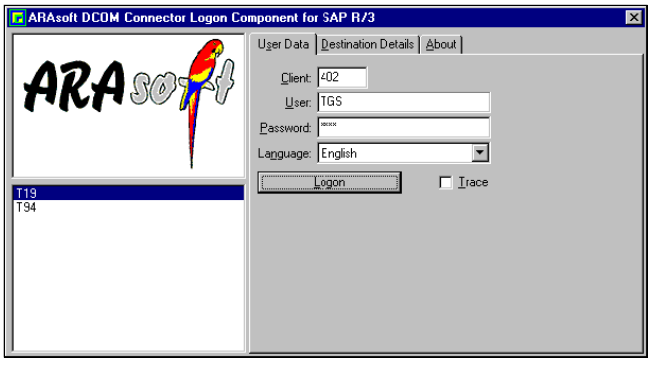
Building a Visual Logon Component

The current version of SDC does not have a dialog component that would allow users to pick a destination from the list stored in the Registry and log on by providing their password and other user information. Based on the Destinations component, it was not too hard to build one. My visual logon component can be downloaded from:

http://www.sap.com/products/techno/bapis/com/ara_dcom_2.zip

and it is freeware. **Figure 13** shows a screenshot of the component.

Figure 13 The ARAsoft DCOM Connector Logon Component



Users logging on to R/3 have to enter a language code unless they want to use whatever default language the system administrator has defined for the particular system. Forcing the user to remember these codes is not nice. My component therefore contains a dropdown combobox, which lists the names of the languages in the languages themselves, so anyone can easily find their own language in the list, instead of having to know the code for the language.

In addition, the component allows the user to turn on the trace without having to change the trace setting for the destination. This allows us to enable the trace for a specific execution of one application without having to turn it on for the destination globally. How is this accomplished? Earlier in this article, I introduced the CreateConnectionString method available for the Destination class. This method retrieves all the information available for the destination and allows the client program to override the client, userid, password, language, and trace default settings, generating the explicit curly braced string that the PutSessionInfo method accepts instead of the destination name.

Using the Logon Component

The file name of the component is `ARAsoft Logon Component.dll`. The name of its Type Library is `ARAsoftDLC`. The component has one global method, called LogonDialog:

```
Function LogonDialog(Session As RfcSession,
                     ErrorMessage As String)
         As LOGONDIALOGRESULT
```

You have to pass a valid Session object and a string that after the call contains the error message returned by the Logon method. The LOGONDIALOGRESULT enumeration contains the constants shown in **Figure 14**.

Figure 14 Constants in LOGONDIALOGRESULT

Const Cancel = 2
Const Failure = 1
Const Success = 0

Listing 4: Invoking the LogonDialog Method

```
Dim oSession As SessionComponent
Dim sErrorMessage As String
Dim ldSuccess As ARAsoftDLC.LOGONDIALOGRESULT
  Set oSession = New SessionComponent
  ldSuccess = ARAsoftDLC.LogonDialog(oSession, sErrorMessage)
  If ldSuccess = Success Then
    MsgBox "Okay"
  ElseIf ldSuccess = Failure Then
    MsgBox sErrorMessage
  ElseIf ldSuccess = Cancel Then
    ' User cancelled the logon dialog
  End If
```

The complete source code required to call the LogonDialog method is given in **Listing 4**.

Once more you can see the benefits of proper encapsulation. Using the component in a client application is extremely easy and straightforward.

Additional Ideas

You have just seen how to use an encapsulation of the CCRegistry API as the basis upon which to build the destination management application and the visual logon component. What else could you do with this Destinations component? You could, for example, build a utility that enables the export and import of XML files containing the attributes of destinations so that an administrator could automatically update the destinations in the Registry of multiple client PCs. This utility could be made part of the startup script for all PCs. When executed, it would fetch the latest version of the XML file from a server and update the local Registry accordingly.

Getting Information About a Session at Runtime

Once a user has successfully logged on to R/3, your application may need to retrieve the following types of session-related information:

- What release of R/3 is running on the server? This information allows your application to use more advanced features available in a higher release of R/3 if, and only if, the user logged on to a system with a certain minimum release level.
- Which userid did the user of your application use to log on?
- What release of SDC is your application running with? (Future versions of SDC will include additional capabilities. An application needs to check the release of SDC before trying to use these capabilities.)
- Which code pages are being used on the client and on the server? (The RFC layer does a good job of converting different data representations, so I have not had to rely on information about code pages, yet. But you might be able to benefit from it somehow.)
- Which language did the user log on with?
- Which character is used as the decimal symbol in R/3?
- Which date format is used by R/3?
- Is the user still connected? (A user could have been disconnected by R/3 after exceeding the

Figure 15 Useful Attributes of GetConnectionAttribute

Attribute Name	Description
OWN_CODEPAGE	Code page on the client
OWN_REL	Release of SDC
PARTNER_CODEPAGE	Code page on the application server
PARTNER_HOST	Application server
PARTNER_REL	R/3 release on the server
SYSID	Name of the R/3 system
SYSTNR	System number
USER	Userid

Figure 16 The Properties of the ConnectionAttributes Class

Property Name	Data Type	Description
SystemId	String	Name of the R/3 system
HostName	String	Application server
UserId	String	System number
SystemNumber	Integer	Userid
SapRelease	String	R/3 release on the server
SapCodePage	String	Code page on the application server
SdcRelease	String	Release of SDC
LocalCodePage	String	Code page on the client

maximum inactivity threshold.) If, for example, a user gets disconnected in the middle of a transaction containing multiple update BAPI calls, the application needs to take appropriate action (e.g., start from scratch).

The first four questions can be answered by calling the Session's GetConnectionAttribute method:

```
Function GetConnectionAttribute
        (Optional AttrName As String)
        As Variant
```

The AttrName parameter allows you to retrieve an individual attribute. When you specify this parameter, the method returns a Variant with the requested data. If you omit the parameter, you get a Recordset with all attributes. The most useful attributes are shown in **Figure 15**.

There are some other attributes in the Recordset that sound promising, like CALLS, LAST_FUNCTION, and STATE, but at least on my computer (using SDC 4.6A), they were either never set (CALLS returned "0", LAST_FUNCTION an empty string) or always contained the same value (STATE always contained "connecting," even after I had been disconnected by R/3).

Encapsulating the GetConnectionAttribute Method

In order to facilitate a client program's access to the important attributes returned by GetConnectionAttribute, I have created a class called ConnectionAttributes.

The properties of this class are shown in **Figure 16**.

The source code for this class appears in **Listing 5**.

After setting the ObjectFactory's Session property, the client program can get an instance of the ConnectionAttributes class by calling the ObjectFactory's GetConnectionAttributes method, shown in **Listing 6**. Then the client program can easily access the various properties of ConnectionAttributes.

Checking the SAP Connection

The only way to find out whether or not a connection to R/3 is still intact is a BAPI or other RFM. The cheapest RFM in terms of performance to call is RFCPING. Therefore, the ObjectFactory contains a method, IsConnectionValid, that tries RFCPING and returns True if that call was successful, False

Listing 5: The ConnectionAttributes Class

```
Private sSystemId As String
Private sHostName As String
Private sUserId As String
Private iSystemNumber As Integer
Private sSapRelease As String
Private sSapCodePage As String
Private sSdcRelease As String
Private sLocalCodePage As String

Public Property Get SystemId() As String
  SystemId = sSystemId
End Property
Friend Property Let SystemId(ByVal sNewValue As String)
  sSystemId = sNewValue
End Property

Public Property Get HostName() As String
  HostName = sHostName
End Property
Friend Property Let HostName(ByVal sNewValue As String)
  sHostName = sNewValue
End Property

Public Property Get UserId() As String
  UserId = sUserId
End Property
Friend Property Let UserId(ByVal sNewValue As String)
  sUserId = sNewValue
End Property

Public Property Get SapRelease() As String
  SapRelease = sSapRelease
End Property
Friend Property Let SapRelease(ByVal sNewValue As String)
  sSapRelease = sNewValue
End Property
```



```
Public Property Get SystemNumber() As Integer
  SystemNumber = iSystemNumber
End Property
Friend Property Let SystemNumber(ByVal iNewValue As Integer)
  iSystemNumber = iNewValue
End Property

Public Property Get SapCodePage() As String
  SapCodePage = sSapCodePage
End Property
Friend Property Let SapCodePage(ByVal sNewValue As String)
  sSapCodePage = sNewValue
End Property

Public Property Get SdcRelease() As String
  SdcRelease = sSdcRelease
End Property
Friend Property Let SdcRelease(ByVal sNewValue As String)
  sSdcRelease = sNewValue
End Property

Public Property Get LocalCodePage() As String
  LocalCodePage = sLocalCodePage
End Property
Friend Property Let LocalCodePage(ByVal sNewValue As String)
  sLocalCodePage = sNewValue
End Property
```

Listing 6: The GetConnectionAttributes Method in the ObjectFactory Class

```
Public Function GetConnectionAttributes() As ConnectionAttributes
Dim rsX As Recordset
Dim caX As ConnectionAttributes
  If oSession Is Nothing Then
    Err.Raise ERR3001, ERRSOURCE, ERR3001Text
  End If
  Set caX = New ConnectionAttributes
  Set rsX = oSession.GetConnectionAttribute
  rsX.MoveFirst
  caX.HostName = rsX("PARTNER_HOST")
  caX.SapRelease = rsX("PARTNER_REL")
  caX.SystemId = rsX("SYSID")
  caX.SystemNumber = rsX("SYSTNR")
  caX.UserId = rsX("USER")
  caX.LocalCodePage = rsX("OWN_CODEPAGE")
  caX.SapCodePage = rsX("PARTNER_CODEPAGE")
  caX.SdcRelease = rsX("OWN_REL")
  Set GetConnectionAttributes = caX
End Function
```

Listing 7: The IsConnectionValid Method in the ObjectFactory Class

```
Public Function IsConnectionValid() As Boolean
  On Error Resume Next
  oFunctions.Rfcping
  If Err.Number <> 0 Then
    Err.Clear
    IsConnectionValid = False
  Else
    IsConnectionValid = True
  End If
End Function
```

otherwise. The source code for this method is shown in **Listing 7**.

The GetConnectAttributes method gives you a means to extract information about the attributes listed in Figure 15. For information about R/3's date format and decimal sign, as well as the language the user logged on with, we turn to a very useful RFM, Rfc_Get_Sap_System_Parameters. This R/3 function returns the parameters shown in **Figure 17**.

Some of this information is already available via a call to GetConnectionAttribute (R/3 release, userid), but the rest cannot be retrieved without calling the Rfc_Get_Sap_System_Parameters function. As before, I have encapsulated the access to this function in the ObjectFactory. The source code for the methods of the ObjectFactory that use the Rfc_Get_Sap_System_Parameters function is shown in **Listing 8**.

Figure 17 The Fields Returned by Rfc_Get_Sap_System_Parameters

Parameter Name	Description
Date_Format	Date format used in R/3
Decimal_Sign	Decimal symbol used in R/3
Language	Logon language
Sap_System_Release	R/3 release on the server
User_Name	Userid

(text continues on page 50)

Listing 8: Additional Methods in the ObjectFactory Class

```
Public Function GetDateFormat() As String
  On Error GoTo EH
  If gsDateFormat = "" Then
    oFunctions.Rfc_Get_Sap_System_Parameters _
      Date_Format:=gsDateFormat, _
      Decimal_Sign:=gsDecimalSign, _
      Language:=gsLanguage, _
      Sap_System_Release:=gsSapRelease, _
      User_Name:=gsUserId
  End If
  GetDateFormat = gsDateFormat
  Exit Function
EH:
  Err.Raise Err.Number, ERRSOURCE, ERRSOURCE + _
    ".GetDateFormat (Line " + CStr(Erl) + ")" + _
    vbCrLf + Err.Description
End Function
```



```
Public Function GetDecimalSign() As String
  On Error GoTo EH
  If gsDecimalSign = "" Then
    oFunctions.Rfc_Get_Sap_System_Parameters _
      Date_Format:=gsDateFormat, _
      Decimal_Sign:=gsDecimalSign, _
      Language:=gsLanguage, _
      Sap_System_Release:=gsSapRelease, _
      User_Name:=gsUserId
  End If
  GetDecimalSign = gsDecimalSign
  Exit Function
EH:
  Err.Raise Err.Number, ERRSOURCE, ERRSOURCE + _
    ".GetDecimalSign (Line " + CStr(Erl) + ")" + _
    vbCrLf + Err.Description
End Function

Public Function GetLogonLanguage() As String
  On Error GoTo EH
  If gsLanguage = "" Then
    oFunctions.Rfc_Get_Sap_System_Parameters _
      Date_Format:=gsDateFormat, _
      Decimal_Sign:=gsDecimalSign, _
      Language:=gsLanguage, _
      Sap_System_Release:=gsSapRelease, _
      User_Name:=gsUserId
  End If
  GetLogonLanguage = gsLanguage
  Exit Function
EH:
  Err.Raise Err.Number, ERRSOURCE, ERRSOURCE + _
    ".GetLogonLanguage (Line " + CStr(Erl) + ")" + _
    vbCrLf + Err.Description
End Function

Public Function GetSapRelease() As String
  On Error GoTo EH
  If gsSapRelease = "" Then
    oFunctions.Rfc_Get_Sap_System_Parameters _
      Date_Format:=gsDateFormat, _
      Decimal_Sign:=gsDecimalSign, _
      Language:=gsLanguage, _
      Sap_System_Release:=gsSapRelease, _
      User_Name:=gsUserId
  End If
  GetSapRelease = gsSapRelease
  Exit Function
EH:
```

```
    Err.Raise Err.Number, ERRSOURCE, ERRSOURCE + _
      ".GetSapRelease (Line " + CStr(Erl) + ")" + _
      vbCrLf + Err.Description
  End Function

  Public Function GetUserId() As String
    On Error GoTo EH
    If gsUserId = "" Then
      oFunctions.Rfc_Get_Sap_System_Parameters _
        Date_Format:=gsDateFormat, _
        Decimal_Sign:=gsDecimalSign, _
        Language:=gsLanguage, _
        Sap_System_Release:=gsSapRelease, _
        User_Name:=gsUserId
    End If
    GetUserId = gsUserId
    Exit Function
  EH:
    Err.Raise Err.Number, ERRSOURCE, ERRSOURCE + _
      ".GetUserId (Line " + CStr(Erl) + ")" + _
      vbCrLf + Err.Description
  End Function
```

As you review the source code you will notice that Rfc_Get_Sap_System_Parameters is called only once, even if the client makes multiple calls to the appropriate ObjectFactory methods. This is accomplished by storing the retrieved information in global variables.

Summary

Congratulations! We have covered a lot of ground together. You should now have all the information required to deal with destination and session management in SDC. If you have any questions or suggestions concerning the presented topics, I would love to hear from you.

Thomas G. Schuessler is the founder of ARAsoft, a company offering products, consulting, custom development, and training around the world. The company specializes in integration between SAP and non-SAP components and applications. ARAsoft offers various products for BAPI-enabled programs on the Windows and Java platforms. These products facilitate the development of desktop and Internet applications that communicate with R/3. Thomas is the author of SAP's CA925 "Programming with BAPIs in Visual Basic" class, which he teaches in Germany and in English-speaking countries. His book on the same subject will be published soon. Thomas is a regularly featured speaker at SAP's TechEd and SAPPHIRE conferences. Prior to founding ARAsoft in 1993, he worked with SAP AG and SAP America for seven years. Thomas can be contacted at thomas.schuessler@sap.com or at arasoft@t-online.de.

