wrox conferences

Building Your Own In-Memory Data Cache

Tim McCarthy, InterKnowlogy

Introduction

A common issue with most WinDNA applications is that they usually need extremely fast access to data that is frequently used. A very reliable way of getting the data is to make a call to the database server every time you need it, and if you are using stored procedures, this process is speeded up a little bit. However, the problem still remains that you have to make a network round trip every time you need access to your data, and in doing so you are probably consuming precious resources on one of your database server machines and also creating network traffic. I am going to address the need for extremely fast, read-only data access in today's WinDNA applications. There are several solutions to this problem; after all, the idea of sharing and accessing data has been around for a long time.

In this paper I am going to discuss building a custom in-memory database (IMDB) to solve this problem. We will implement a custom IMDB with a fairly common scenario and a sample application. This article is targeted at the intermediate to advanced Visual Basic (VB) programmer, and assumes that the programmer is familiar with COM+ and WinDNA. Please note that this paper will not address the problem of state management. That is an entirely separate issue.

Tim McCarthy is a Principal Engineer at InterKnowlogy (www.InterKnowlogy.com), where he architects and builds highly scalable n-tier web applications utilizing the latest Microsoft technologies. He is an author and technical reviewer for Wrox Press, working most recently on Professional ADO 2.5 Programming. Tim is a regular speaker at Microsoft Developer Days, and is also an instructor at the University of California, San Diego, where he teaches Microsoft Certified Solution Developer (MCSD) courses.

Sharing Data

There are many ways in WinDNA applications to store, access, and share data. I am going to assume that most people who are implementing a WinDNA application are using SQL Server as their relational database management system (RDBMS). The idea of repeating calls to a database over and over for retrieving infrequently changing data will ultimately affect both the performance and scalability of your application. Consider the common scenario where a list box on a web page has a listing of all the states. Instead of retrieving the data for the states and formatting the HTML on every call to the page, a better solution is to cache the list box HTML in a global location so the web server does not have to retrieve the data and render the HTML every time that the page is requested. A good way to store this data would be as a string inside an ASP application variable that is initialized in the `Global.asa` file. In fact, the application variable is a good place to store global data on a web server – it is both efficient and fast. But what happens if you need to cache the data for other applications, not just an IIS application? What if you have multiple IIS applications, running on the same machine, that need to share data? One solution is store data locally on the web server in such a way that all applications running on the server can access the data.

Here are some of the many ways for storing data locally:

- Have SQL Server locally installed on your web server and replicate between the master and replicated database servers
- Use memory-mapped files
- Use the Shared Property Manager
- Use the Global Interface Table in COM

Local SQL Server Installation

It is entirely possible to have SQL Server locally installed on your web server and replicate between the master and replica database servers for data that is frequently looked up. However, this defeats the purpose of allowing the web server to handle more requests, because SQL Server demands a *lot* of resources on a machine in order to operate efficiently. These are resources that the web server desperately needs to use for serving up and processing web pages. The most common architecture scenario for a large WinDNA application is to have one or more web servers that are load balanced (using Windows Load Balancing Service), and have them pointing to one virtual SQL Server machine that is using Microsoft Failover Support for SQL Server and the Windows Clustering Service. For more information on this, see http://msdn.microsoft.com/library/psdk/sql/1_server_17.htm. As a result, it is not normally possible to replicate the SQL database on each node in the web server farm.

> **By using this technique, the web servers are "scaled out" and the SQL Servers are "scaled up." Scaling out means adding more machines to handle the distributed load, and scaling up means increasing the hardware on a machine to handle a load.**

Memory-Mapped Files

Memory-mapped files allow a file on disk to be associated with an address space in memory. Once this mapping is done the data in the file can be accessed as if the file was in memory. Memory-mapped files provide a method in Windows for sharing blocks of memory between processes. Although this seems like a great solution, in VB it involves several calls to the `CopyMemory` API function as well as several other API functions, such as `CreateFileMapping`, `OpenFileMapping`, `MapViewOfFile`, `MapViewOfFileEx`, `UnmapViewOfFile`, `FlushViewOfFile`,

and `CloseHandle`. To access the content of a memory-mapped file, you must de-reference a pointer in the designated range of addresses. So, writing data to a file is accomplished by assigning a value to a de-referenced pointer. One reason for not choosing this solution is that we would have to store our data as some type of flat file, or possibly an XML string; every time we need to open the data we would have to either use the MS DOM COM object or use an ADO Recordset object. This process of constantly opening the data could slow down our application. Another reason for not choosing this solution is that we would have to write some plumbing code to control simultaneous access to the data. Note that VB was never intended to deal with pointers in a graceful way and, with the upcoming Visual Studio .NET and the Common Language Runtime, we will not be using pointers any more. Therefore, memory-mapped files are best implemented in C++.

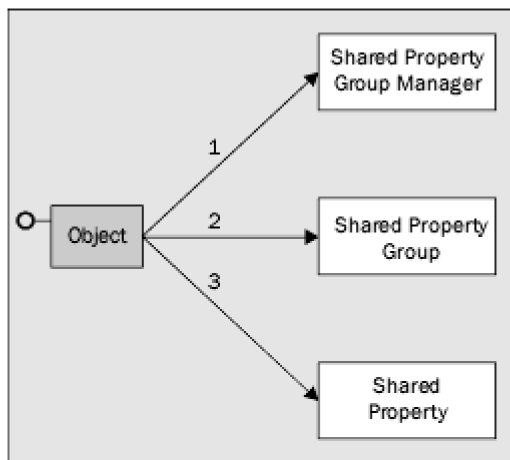
The Shared Property Manager

The Shared Property Manager (SPM) is a resource dispenser that you can use to share state among multiple objects within a server process. The SPM implements locks and semaphores to protect shared properties from simultaneous access.

> **Semaphores are protected data objects used for communication with other processes in Windows.**

Because of this, it is not a good idea to store large amounts of data or COM objects in the SPM, as it will cause serialized access to the data and will seriously hamper the performance and scalability of your application. The idea is to store small amounts of data, such as integers. This way, any one user does not tie up the SPM for a long period of time.

The SPM is composed of a fairly simple object model, with the root level being the Property Group Manager object. Under the Property Group Manager object is the Property Group object, and under this is the Shared Property object. Each object has its own name space. The Shared Property object is where the data is stored and retrieved via the `Value` property. See the diagram below for a picture of what I am describing here.



Shared properties stored in the SPM are only available to objects running in the same process. This means that the objects that will use the SPM for storing values, and that will need to have access to these values, must be installed as part of the same COM+ application. If they are not in the same COM+ application they will not be in the same process, and therefore will not be able to access the values. It's also important for components sharing properties to have the same activation attribute. If two components in the same package have different activation attributes, they generally won't be able to share properties. For example, if one component is

configured to run in a client process and the other is configured to run in a server process, their objects will usually run in different processes, even though they are in the same package.

The Global Interface Table

The Global Interface Table (GIT) was introduced in Windows NT 4.0 SP3 and it allows interface pointers to be accessed by all apartments in the same process. COM internally implements one GIT per process. The GIT contains marshaled interface pointers that can be efficiently unmarshaled multiple times within the same process. The pointer can be a pointer to an in-process object, a proxy for an object residing in another apartment, in another process, or in another machine. The GIT is only accessible through C++; however, the legendary Don Box wrote a VB wrapper around the GIT that allows you to store, retrieve, and release references from the GIT (see http://www.develop.com/dbox). The way that the GIT works is that, when you register an interface into the GIT, it returns a cookie value (a `DWORD` in C++ or a `Long` in VB) that you can use to retrieve the same interface again out of the GIT. The same cookie is also used to remove the interface from the GIT. This allows us to store pointers to COM objects that can contain data structures such as ADO Recordset objects, without having to initialize the objects before using them. This saves both time and resources.

A Hybrid Approach

Another approach is to combine some of the ways discussed previously for storing data in memory. In my chosen solution, I have decided to use the GIT to hold a reference to a COM object that contains ADO Recordset objects that are accessible via its methods. I am also going to use the SPM to hold the cookie value of the COM object that the GIT produced. This will allow us to take advantage of the GIT's ability to eliminate unmarshaling interface pointers multiple times between apartments in the same process. At the same time, we are also taking advantage of the SPM's ability to share transient state between objects in the same process. Since we are storing small values (VB `Long` datatypes), we should not have any locking or contention issues when getting the values in and out of the SPM.

We are also going to use the SPM to manage the locking and unlocking of the COM object, to allow for updates to the COM object's data. All we have to do to implement this is to add new properties to the SPM to keep track of which set of data is dirty or not, basically a Boolean flag. Then, before updating the data, we just set the dirty flag(s) to true, do our update, and then set the dirty flag(s) to false. This will prevent multiple threads from stepping on each other when the data needs to be updated in the COM object. One very important thing to remember in this approach is that all of the components need to be running in the same COM+ application. This will ensure that all requests for data will be in the same process ID as the COM object's instance and the SPM's instance.

This approach will be clearly demonstrated in the sample application code.

Data Synchronization

Previously, we briefly touched on updating the data in the cache. This is a very important part of the solution, because no matter how fast access to the data is, it does us no good if it the data is not current. There are some questions that you should be asking yourself. What happens when someone updates data in the SQL Server database – how do we know to update the local cache? How often do we check for updates? The first thing that comes to mind when I think about this is that we have to control access to the database. This isn't so hard to do; we just need to make sure that all access to the database goes through a COM object. By doing this we have control over when the data is being updated. We could change the implementation of the COM object to write out data to a table to indicate what has been updated during one of the update, insert, or delete actions. The question, however, still remains – how do we, as a lonely data cache on a web server, get notified about the change in

the data? There are two main ways to do this. We can do it passively, with a service running on our web server that continually polls the database for changes, or we can implement some kind of event system. If we decide to go down the passive route, we would be creating extra network traffic and also eating up precious resources on the SQL Server machine. Therefore, it would be better to use an event system, and the answer is COM+ Events.

COM+ Events

COM+ Events is a Loosely Coupled Event (LCE) system that stores event information from different publishers in an event store in the COM+ catalog. Subscribers can query this store and select the events that they want information about. Querying the store can be done via the Component Services administration tool or it can be done programmatically. Selecting event information from the event store creates a subscription. When an event occurs, the event system checks the event store database to find the interested subscribers, creates a new object of each interested class, and then calls the appropriate method on each subscribing object.

To create an event, you need to make what is called an `EventClass` – a COM+ component that declares the interface used by a publisher to fire events. The subscriber must implement these interfaces in order to receive events. To do so, you need to specify the interfaces and methods you want an `EventClass` to contain. This is easy to do in VB; you just compile an ActiveX DLL project that has classes and methods, and do not put any code in the methods. This will be described in more detail later. `EventClasses` are entered into the COM+ event store by publishers using the Component Services administration tool or via code.

In the case of a web server holding a cache of data in memory, the best way to implement the data synchronization is to designate one machine on the network as the centralized COM+ Event store, within a cluster to avoid a single point of failure. This can be done with Application Center 2000 or with the Windows Clustering Service (see http://www.microsoft.com/applicationcenter). This machine will hold all of the `EventClasses` and their subscriptions. The data cache COM object on the web server would then need to be set up as a subscriber to one or more methods in the `EventClass`. Any time data needs to be updated in the SQL Server database, it should be done via a COM interface to the stored procedures. In the method calls to the update of the database, the only thing that needs to be done to fire the event is to simply instantiate the `EventClass` and then call one of its methods.

The Sample Application

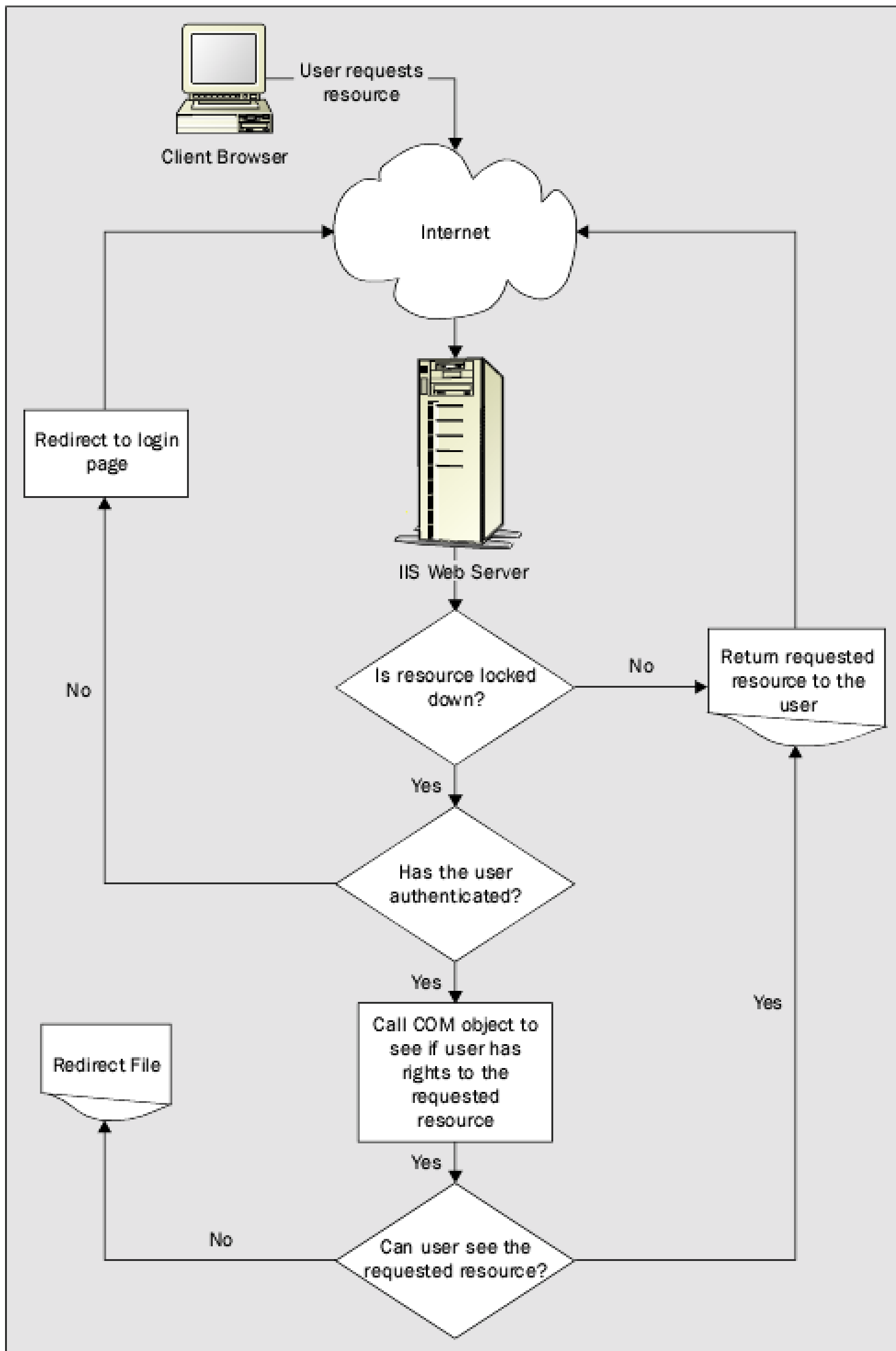
In order to demonstrate all of the functionality of the custom IMDB and the use of COM+ Events for synchronizing the data, we will be looking at a sample web application.

Scenario

The scenario for our sample application will be a web site that wants to maintain its own security instead of using Windows NT security for their content. They want to establish their own database of users and groups that are associated with resources or web content. The idea is to store user names and passwords in a SQL Server database rather than creating NT accounts for each user. Using this approach will allow a properly architected web site to support an almost unlimited number of registered users. Site Server 3.0 Personalization and Membership takes the same type of approach to solving this problem.

The site should first dynamically determine if a user is trying to access a resource that has been locked or not. If the user is trying to view a locked resource, then the site should determine if they have been authenticated or not. If the user has not been authenticated, then they should be redirected to a login page. If the user has been authenticated, the site will check to see if they have permission to view the locked resource or not. If the user does not

have permission, then they will be redirected to a designated page explaining that the resource could not be accessed. Below is a diagram of the authentication process:

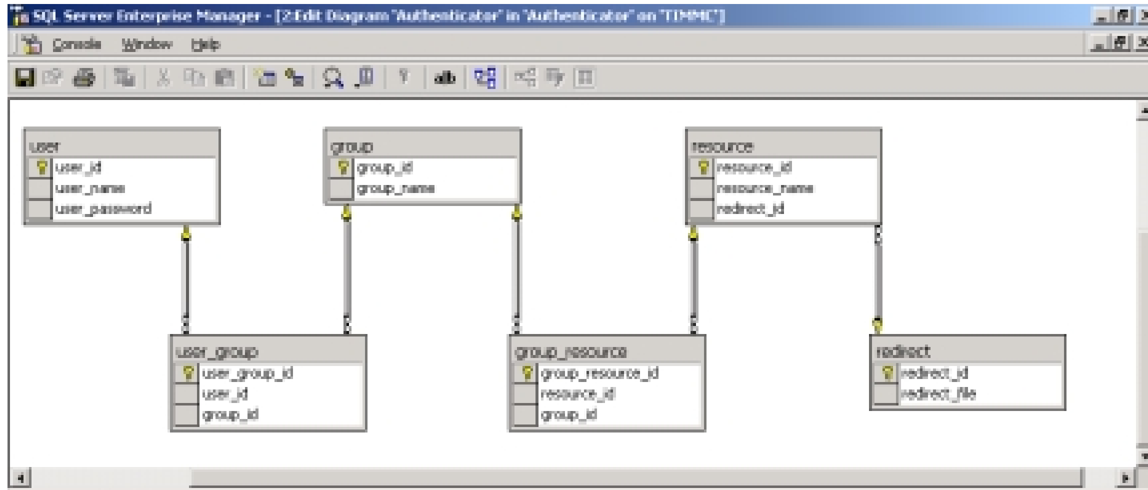


Solution/Application Architecture

Database

All resources will be locked via our database instead of using Windows NT Access Control Lists (ACL) and Access Control Entities (ACE). This will allow us more flexibility in how the files are locked. For instance, using ACLs and ACEs, we would only be able to lock resources by users

and their groups. If we use a database to lock resources, we then have the added flexibility of choosing other means of locked resources besides just users and groups. If we wanted to, we could lock resources down with complex security rules and profiles, which is not very elegant using ACLs and ACEs. The following screenshot shows a diagram of the database for our application:



Custom IMDB Object

Because we do not want to make network trips to our database server for authentication purposes, we must keep the security data cached locally on the web server. In order to support our business requirements, we will transform the data in the tables of our database into three logical structures – security, users, and resources. The security data structure will consist of resources and their associated users, so we can determine if users have rights to view a requested resource. We need to maintain a structure of users in memory so we can verify users when they log in to the site. The resources data structure will be used so we know what resources are locked or not. We will be using ADO Recordset objects in memory to store this information.

The way we will implement the custom IMDB is by using the hybrid approach that we looked at earlier which involved both the GIT and the SPM. All of the access to the data in the IMDB will be encapsulated in one VB ActiveX DLL, `WCache.dll`. This DLL will contain one class, the DataCache class, which is the interface for all data access to the IMDB. It contains a pair of `Get` and `Set` functions for each logical data structure (security, users, and resources).

Below is the code for the class declarations, and the `Class_Initialize` and `Class_Terminate` events.

```
Option Explicit

'The data sets in our cache
Private mrstResources As ADODB.Recordset
Private mrstSecurity As ADODB.Recordset
Private mrstUsers As ADODB.Recordset

'The Shared Property Manager variables
Private mspmMgr As SharedPropertyGroupManager
Private mspmGroup As SharedPropertyGroup
Private mspmSecurityDirty As SharedProperty
Private mspmResourceDirty As SharedProperty
Private mspmUsersDirty As SharedProperty
Private mblnExists As Boolean
```

```
Private Const MODULE_NAME = "DataCache"

'Shared Property Manager constants
Private Const LOCK_SET_GET = 0
Private Const LOCK_METHOD = 1
Private Const STANDARD = 0
Private Const PROCESS = 1

'Win API function declaration
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds _
     As Long)

'The COM+ Event class we are subscribing to
Implements WCacheEvents.Data
```

```
Private Sub Class_Initialize()

'This method will get fired when this class is instantiated. The
'purpose of this code is to initialize Shared Property Manager,
'create a Property Group and add some properties. The group name
'is "DATA_CACHE" and the three properties are "IsSecurityDirty",
'IsResourcesDirty", and "IsUsersDirty". These groups correspond
'directly with our three sets of data in the cache, Security,
'Resources, and Users.

On Error GoTo ErrorHandler

     'Instantiate the SharedPropertyGroupManager object
     Set mspmMgr = New SharedPropertyGroupManager

     'Use the SharedPropertyGroupManager object to get the existing
     'PropertyGroup (DATA_CACHE). If the group already exists, it will not
     'be created again, it will just be retrieved. If the group does not
     'exist, it will be created. Use LOCK_SET_GET so we don't lock all of
     'the properties in the group, only the current property that we are
     'accessing. Use PROCESS to make sure the property group stays open
     'until our process has terminated.
     Set mspmGroup = mspmMgr.CreatePropertyGroup("DATA_CACHE", _
          LOCK_SET_GET, PROCESS, mblnExists)

     'Use the PropertyGroup object to create the three properties.
     Set mspmSecurityDirty = _
          mspmGroup.CreateProperty("IsSecurityDirty", mblnExists)
     Set mspmResourceDirty = _
          mspmGroup.CreateProperty("IsResourcesDirty", mblnExists)
     Set mspmUsersDirty = _
          mspmGroup.CreateProperty("IsUsersDirty", mblnExists)

ExitProc:
     Exit Sub

ErrorHandler:
     Err.Raise 700007, MODULE_NAME & ".Class_Initialize", _
          Err.Description
     Resume ExitProc

End Sub
```

```
Private Sub Class_Terminate()

     'Clean up

     'The data
     Set mrstResources = Nothing
     Set mrstSecurity = Nothing
     Set mrstUsers = Nothing

     'The spm
     Set mspmMgr = Nothing
     Set mspmGroup = Nothing
     Set mspmSecurityDirty = Nothing
     Set mspmResourceDirty = Nothing
     Set mspmUsersDirty = Nothing
End Sub
```

We start by declaring our three module level ADO Recordset objects that contain the security, user, and resource data previously mentioned. We also declare a variable to hold the `SharedPropertyGroupManager` object (`mspmMgr`) and a variable to hold the `SharedPropertyGroup` object (`mspmGroup`). The purpose of `mspmMgr` is just to get a handle to the SPM so we can create a property group. We then create our property group via the `CreatePropertyGroup` method of `mspmMgr`. We then use the `CreateProperty` method of `mspmGroup` so we can create all of our shared properties. Note how all of the "`Create`" methods of the SPM objects are being passed a Boolean variable (`mblnExists`). This is an in/out parameter that will be set to `True` if the group or property already existed. The SPM will not create a second instance of the property group; it will only create a single instance of it. In our case, we do not care what the value of `mblnExists` is; we just need to pass it in because the methods require it.

On the `CreatePropertyGroup` method of `mspmMgr`, the arguments are name, `dwIsoMode`, `dwRelMode`, and `fExists`. The name argument is simply the name of the property group; in our case we are calling it "`DATA_CACHE`".

It is extremely important that the group is accessed by the same name all of the time. If a different name is passed in, then a new group will be created by the SPM.

The `dwIsoMode` argument defines how the group will be locked. `LOCK_SET_GET (0)` will lock the particular property while it is being set, and `LOCK_METHOD (1)` locks all of the properties in the property group until the method is complete. The `dwRelMode` argument specifies how the property is destroyed. `STANDARD (0)` destroys the property group when all of the clients have released their reference to it. `PROCESS (1)` signals that the property group is not destroyed until the process in which it lives is terminated. In our code, we have set the `dwIsoMode` to `LOCK_SET_GET` to allow for the least amount of locking level, and we have set the `dwRelMode` argument to `PROCESS` since we want our group to stay alive as long as possible without having to be re-created all of the time.

We are creating three properties for our group. They are `IsSecurityDirty`, `IsResourcesDirty`, and `IsUsersDirty`. These properties will be used as blocking flags when we update the data in the cache. In the `Class_Terminate` event, we are simply just releasing all of the references to our objects. This should never really be called since our object will be kept alive in the GIT. The only time our object should "die" is when the machine is rebooted.

The next code segment is basically the same for all `Get` and `Set` operations, so I will only show the `GetResources` and `SetResources` methods. Here is the code:

```
Public Sub GetResources(ByRef Resources As ADODB.Recordset)

'This method takes an ADO Recordset object passed in by reference,
'and sets its pointer to the mrstResources module-level recordset.

On Error GoTo ErrorHandler

     Const ProcName = "GetResources"

     'Check to see if we have the data in the cache. If not, then
     'we need to set the cache.
     If mrstResources Is Nothing Then
          Call SetResources
     End If
     Set Resources = mrstResources

ExitProc:
     Exit Sub

ErrorHandler:
     Err.Raise 700001, MODULE_NAME & "." & ProcName, _
          Err.Description
     Resume ExitProc
End Sub
```

```
Public Sub SetResources()

'This method gets the resource data from the SQL Server database
'and populates the module-level variable mrstSecurity.

On Error GoTo ErrorHandler

     Const ProcName = "SetResources"

     'Our loop counter
     Dim intLoopCount As Integer

     'In case the cache is dirty, we will either run this loop 3
     'times, or until the data is not dirty, whichever comes
     'first.
     Do

          'Increment the counter
          intLoopCount = intLoopCount + 1

          'If after the third try we cannot do an update, then
          'raise an error. This should never happen, but just in
          'case...
          If intLoopCount > 3 Then
               Err.Raise 700002, ProcName, "Error trying to " &_
                  "write to the data cache. The cache is locked."
          End If

          'If the data is not dirty
          If mspmUsersDirty.Value = 0 Then
```

```
                'Set the dirty flag to true
                mspmResourceDirty.Value = 1

                'Get the data from the database
                Set mrstResources = _
                     ExecSPReturnRS("usp_Get_Resource_Data", _
                     MakeParameter("RETURN_VALUE", adInteger, _
                     adParamReturnValue, 4))

                'Set dirty flag to false
                mspmResourceDirty.Value = 0

           Else

                'The cache is dirty. Wait 1/2 second and try again
                Call Sleep(500)

           End If

      'Keep looping until the cache is not dirty
      Loop Until mspmResourceDirty.Value = 0

ExitProc:
     Exit Sub

ErrorHandler:
     Err.Raise 700002, MODULE_NAME & "." & ProcName, _
          Err.Description
     Resume ExitProc
End Sub
```

The `GetResources` method takes an ADO Recordset object passed in by reference as its only argument. It then checks to see if the module-level variable `mrstResources` has been instantiated yet or not. If it has not yet been instantiated, a call is made to the `SetResources` method to instantiate and populate the recordset with data. The `SetResources` method has no arguments and its mission is to instantiate and populate the `mrstResources` variable with data from the database server. It first starts out in a loop to see if the `IsResourcesDirty` flag has been set to `True` or not. If it is dirty, it will then pause for a one half second, and try again. If the data is still dirty after three tries, the method raises an error. If the data is not dirty, it will set the dirty flag to `True`, retrieve the data from the database, and then set the flag to `False`. This method calls the `ExecReturnRS` helper function in the `basDatabase` code module to retrieve the data for `mrstResources`.

The next obstacle that we have to overcome in our custom IMDB object is the issue of data synchronization with the SQL Server database. This is done with COM+ Events. Our COM+ Event Class will be an abstract class in the WCacheEvents DLL. There are really three data action activities that we are concerned with in this application, and they occur when resources, security, or users have been added, updated, or deleted. This equates to three operations that we care about – when resources change, when security changes, and when users change. Therefore, here is the code for our COM+ Event Class, WCacheEvents.DataCache:

```
Public Sub ResourcesChanged()

End Sub

Public Sub SecurityChanged()
```

```
End Sub

Public Sub UsersChanged()

End Sub
```

To register the event class in COM+, all you need to do is go back to the COM+ application (Authenticator) where our other COM objects live, and add a new component. You will select Install new event class(es) and the wizard will guide you through the rest of the way.

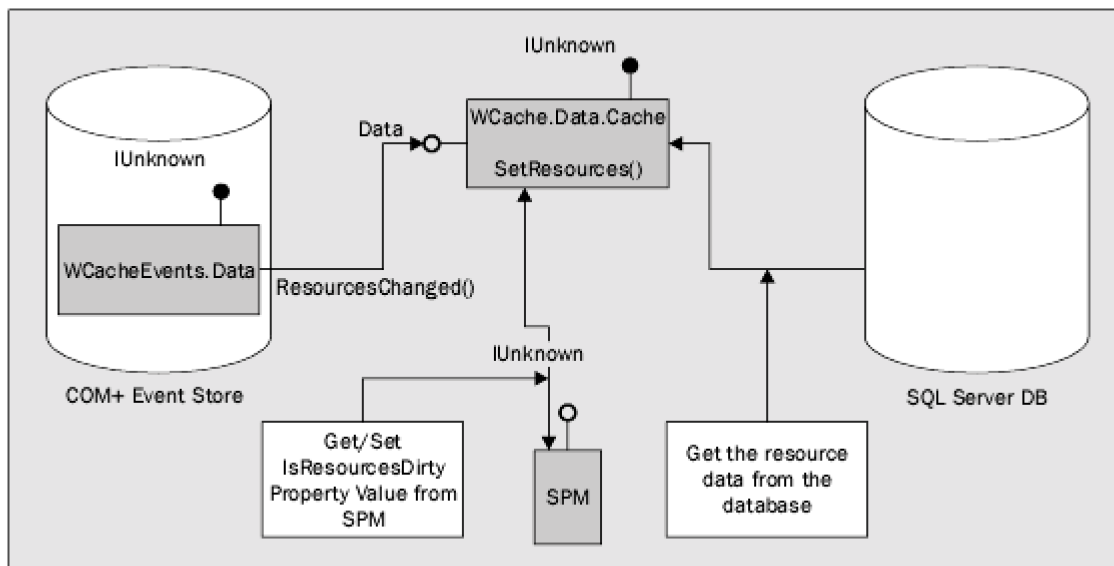
The next thing we need to do after registering our event class is to have our custom IMDB object subscribe to it and implement it. This is the same process as implementing any other interface in VB. Here is what the code looks like in the `WCache.DataCache` class:

```
Private Sub Data_ResourcesChanged()
   Call SetResources
End Sub

Private Sub Data_SecurityChanged()
   Call SetSecurity
End Sub

Private Sub Data_UsersChanged()
   Call SetUsers
End Sub
```

Here is a code sample of how to fire off the `ResourcesChanged()` event from another application:

```
Sub FireEvent()

   Dim objEvent As WCacheEvents.Data

   Set objEvent = CreateObject("WCacheEvents.Data")
   objEvent.ResourcesChanged
   Set objEvent = Nothing

End Sub
```

The following diagram gives a graphic view of the architecture and process for updating the cache:



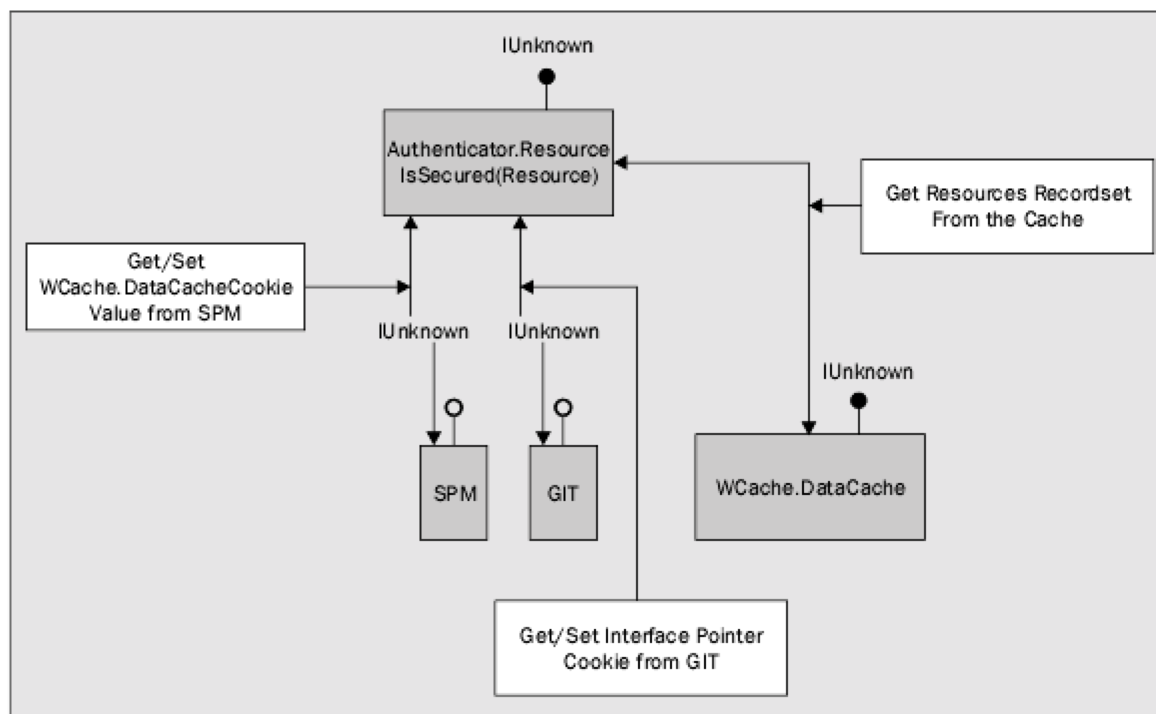
Business Objects

This application involves another ActiveX DLL, `Authenticator.dll`. This DLL holds the business logic for our application and has two classes, Resource and User. The only code that is really worth mentioning here in this object is the code that places our custom IMDB object into the GIT. Each class in `Authenticator.dll` implements the COM+ ObjectControl interface, and the `ObjectControl_Activate` method is where the code is executed that interacts with the GIT. Here is the code for the `ObjectControl_Activate` method:

```
Private Sub ObjectControl_Activate()

'This method will get fired by COM+ when this class is
'instantiated. The purpose of this code is to query the Shared
'Property Manager for the cookie of the WCache.Data interface
'pointer. If the cookie does not exist, that means that the
'interface pointer has not been placed in the GIT yet. In that
'case, we must create the object and place the pointer into the
'GIT. If the cookie value exists, then we use that as our key to
'get our live interface pointer out of the GIT.

On Error GoTo ErrorHandler

     Const ProcName = "ObjectControl_Activate"

     Dim lngCookie As Long
     Dim spmMgr As SharedPropertyGroupManager
     Dim spmGroup As SharedPropertyGroup
     Dim spmCacheCookie As SharedProperty
     Dim blnExists As Boolean

     'Get the current COM+ context
     Set mobjContext = GetObjectContext

     'Instantiate the SharedPropertyGroupManager object
     Set spmMgr = New SharedPropertyGroupManager

     'Use the SharedPropertyGroupManager object to get the
     'existing PropertyGroup (DATA_CACHE). If the group already
     'exists, it will not be created again, it will just be
     'retrieved. If the group does not exist, it will be created.
```

```
     'Use LOCK_SET_GET so we don't lock all of the properties in
     'the group, only the current property that we are accessing.
     'Use PROCESS to make sure the property group stays open
     'until our process has terminated.
     Set spmGroup = spmMgr.CreatePropertyGroup("DATA_CACHE", _
                    LOCK_SET_GET, PROCESS, blnExists)

     'Use the PropertyGroup object to get the existing Property
     '(CacheCookie). If the property already exists, it will not
     'be created again, it will just be retrieved. If it does not
     'exist, it will be created.
     Set spmCacheCookie = spmGroup.CreateProperty("CacheCookie", _
                          blnExists)

     'Get the value of the CacheCookie property
     lngCookie = spmCacheCookie.Value

     'If the cookie is greater than 0, then we do not have to
     'create the WCache.Data object, we simply retrieve it from
     'the GIT. Otherwise, we have to create it and then, after
     'we create it, we put it into the GIT.
     If lngCookie > 0 Then
          Set mobjCache = _
               GITHelpLib.GetInterfaceFromGlobal(lngCookie)
     Else
          Set mobjCache = CreateObject("WCache.Data")
          lngCookie = _
               GITHelpLib.RegisterInterfaceInGlobal(mobjCache)
          spmCacheCookie.Value = lngCookie
     End If

ExitProc:
     Set spmMgr = Nothing
     Set spmGroup = Nothing
     Set spmCacheCookie = Nothing
     Exit Sub

ErrorHandler:

     'Call error handler routine
     Call HandleError(MODULE_NAME, ProcName, Err.Number, _
          Err.Source, Err.Description, , mobjContext)

     'Go to clean-up routine
     Resume ExitProc
End Sub
```

The following diagram gives a graphic view of the architecture and process for retrieving data from the cache:



The first part of this code should look familiar; we are just accessing the Shared Property Manager to get the value of the `CacheCookie` property. This property holds a `Long` value that is the cookie for unlocking the custom IMDB interface out of the Global Interface Table. If this value is zero, then we simply create the WCache.DataCache object and place it into the GIT. If it is greater than zero, then we retrieve the interface from the GIT. The `mobjCache` module variable is an instance of the WCache.Data COM object. The rest of the Resource class has just two methods, `IsSecured` and `IsAccessible`. The `IsSecured` method has one argument, a string containing the name of the resource to query the IMDB, to see if it is secured or not. The `IsAccessible` method has three arguments – the first is a string containing the name of the user, the second is a string containing the name of the resource, and the third is an in/out string parameter containing the name of the redirect file. These arguments are used to query the IMDB to see if a particular user has access to a particular resource or not. The User class only has one method in it, `VerifyUser`. This method has two arguments – a string value containing the username, and a string value containing the associated password. This method queries the IMDB to see if a user with the given username and password exists, and, if the user exists, it will return a value of "`True`".

As you can see, the Authenticator COM objects are fairly simple. They validate users, check if resources are secured, and check to see if users have access to various resources. This is all done by making calls to the Wcache.DataCache COM object, which holds the data we need in memory.

Summary

In this paper we have covered the use of the SPM, the GIT, and COM + Events – and how to use all of them in unison to build your own custom IMDB. There are several other ways to do this, but this methodology was chosen due to the ease of coding and somewhat elegant architecture of using some of the built-in facilities of the Windows 2000 platform. We also looked at a sample application to demonstrate our architecture.

Further Resources

MSDN: http://www.msdn.microsoft.com

Professional Active Server Pages 3.0, Wrox Press (ISBN 1861002610).