Enhancing Membership

DotNetNuke Roadmap

Charles Nurse

Version 1.0.0
Last Updated: June 21, 2006
Category: DotNetNuke 3.3/4.1

Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results of the use of this document remains with the user.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, places, or events is intended or should be inferred.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Perpetual Motion Interactive Systems, Inc. Perpetual Motion Interactive Systems may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Perpetual Motion, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

DotNetNuke® and the DotNetNuke logo are either registered trademarks or trademarks of Perpetual Motion Interactive Systems, Inc. in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

DOTNETNUKE™

Enhancing Membership

Abstract

This document describes proposed enhancements of the Membership capabilities of DotNetNuke.

Contents

Introduction

This document proposes a number of enhancements for membership.

- Single Sign-On/Portal groups – the ability for a User to be a member of multiple portals using the same user credentials (username/password)
- Removal of dependence on HttpContext
- Password enhancements
  - Offer Hashed passwords as an option
  - Password reset by Administrator
  - Implement Question and Answer for password retrieval or reset
  - Enforce Password Complexity (min size, min alphanumeric, regular expression validation)
  - Password Aging/Expiry
  - Force Password Update (or Profile update)
  - Enable/Disable Password Retrieval
  - Enable/Disable Password Reset
- Enhancements for SignIn/Logout
  - CAPTCHA
  - RedirectPage after SignIn (when from Home page)
  - UrlReferrer needs to "retain" querystring params
  - Email administrator after user lockout
  - Redirect after logout (not always to Home)
- Registration Enhancements
  - CAPTCHA
  - Public Registration – send email to user in case the email used is not the user registering
  - Send verification url that links directly to the "SignIn page"
  - Send email on profile change to alert user in case an Unauthorized user has updated their profile

- Integrate UsersOnline – the Users Online API should be a part of the "membership" component
- Demo User Accounts – the ability to create Demo Users with "limited" ability to modify their profile/password.
- Enhanced the "Approved" status to a multi-mode capability, to support Banned and other statuses

Some of these enhancements may require us to consider abandoning the AspNet Membership components altogether. They will definitely have an impact, as some of the Password enhancements would need to be at the host level in the AspNetProvider.

Single Sign-On/Portal Groups

Introduction

There has been a great deal of debate about this feature since the release of DNN 3.0 and the use of the MemberRole components, as this capability was present in DNN 2.0. However, in reality the implementation in DNN was flawed and it was "enforced", ie the host user could not switch it off.

There are three distinct Use-Cases that we should consider in this enhancement:

1. A User's credentials will only work with a single-portal in the Application. In addition if the same username is used to register to a different portal in the Application, the user is either rejected or a completely separate User is created with its own password and profile. What happens here will depend on some implementation issues.

2. A User's credentials will work automatically in all portals within the Application. On registration if this user has already registered with one portal on the site the system will check if the passwords match. If the passwords don't match the user will be rejected. If the passwords do match the user will be given the opportunity to "review" their profile data (rather than automatically updating as was the case in DNN 2.x).

3. A Hybrid of the two other Use Cases: in this situation a User's credentials can be used to sign on to a specified group of Portals, but other portals are not part of the group and so the user's credentials will not work in that situation.

Portal Groups

If we add the concept of Portal Groups as an extra layer we can effectively support all of these scenarios.

In scenario (3) Portals could be assigned to one or more Portal Groups.

Thus if we had two Portal Groups (Group1 and Group2) and 4 Portals (PortalA, PortalB, PortalC and PortalD) then we could assign PortalA and PortalB to Group 1 and PortalB and PortalC to Group2.

Group1	Group2	UnAssigned
PortalA	PortalB	PortalD
PortalB	PortalC	

This grouping shows three possible scenarios.

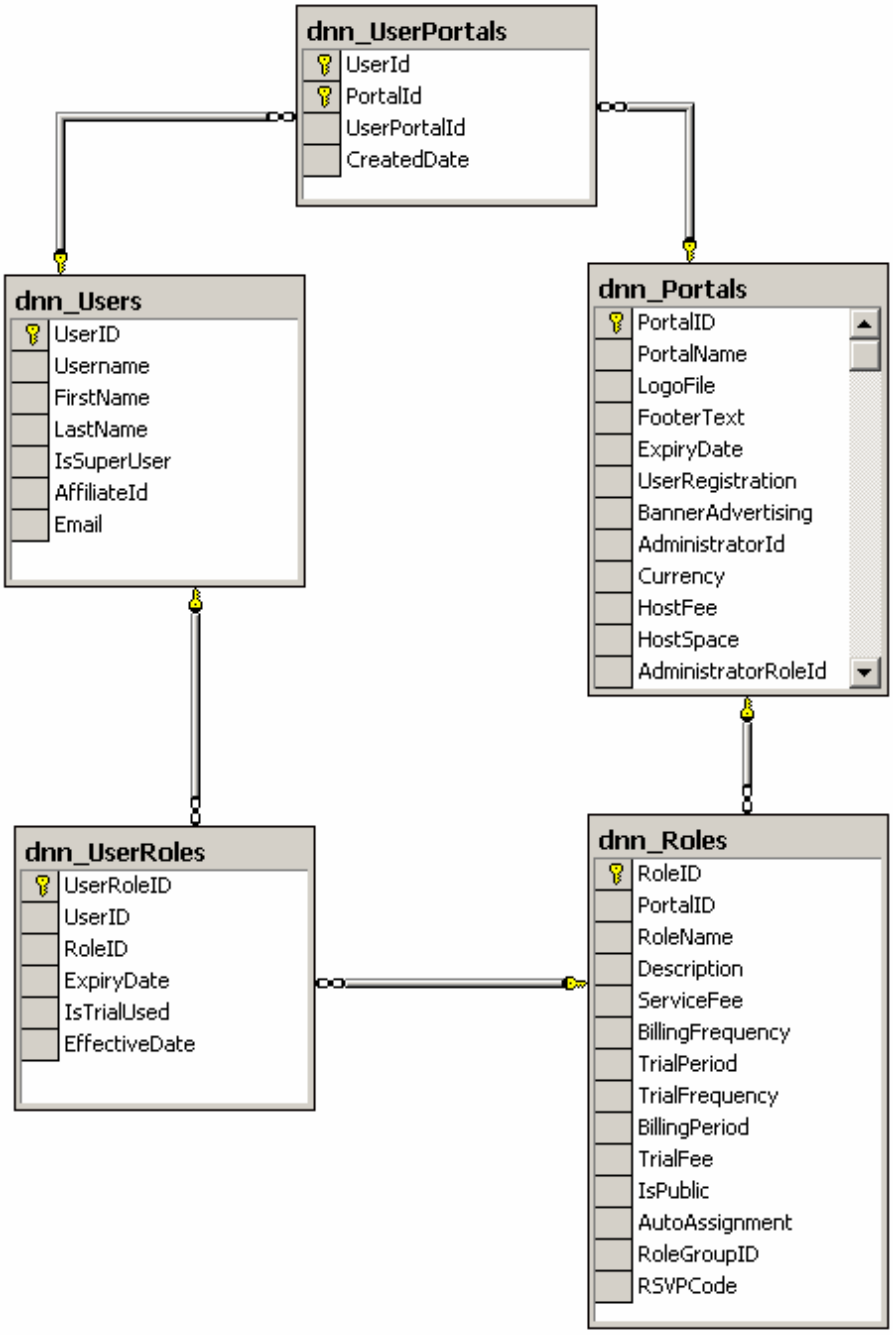
- A user that registers with PortalA would automatically be assigned to Group1 (and therefore also be a member of the related PortalB). Similarly, a user that registers with PortalC would automatically be assigned to Group2 (and therefore also be a member of the related PortalB).
- A user that registers with PortalB would automatically be assigned to both Group1 and Group2 and therefore become members of both PortalA and PortalC.
- A user that registers with PortalD would only be a user in that portal.

In scenario (2) Portals would automatically be assigned to the pseudo-group (AllPortals) and therefore registration in one portal would give the user automatic membership of any portal on the system

In scenario (1) Portals users would be assigned directly to the Portal and only to the Portal, so the user would only have membership to the Portal he/she registered with.

This concept of Portal Groups as described here would allow for the most flexibility. It would require that Users can be assigned directly to portals (eg PortalD), directly to a single Portal Group (PortalA -> Group1, PortalC -> Group2) or to multiple Portal Groups (PortalB -> Group1, Group2).

Current Situation



The previous discussion regarding PortalGroups essentially is equivalent to saying "A user can be a member of many Portals. A Portal can have many Users.

The UserPortals table is effectively a many-to-many join table. So could this table not be used to provide Portal Groupings?

Using this scenario would allow us to implement option (2), where a user is a member of all portals. This is because there is no provision in this arrangement for a Portal to be "excluded" from the list of Portals that a user is a member of.

New PortalGroups Table (Info Class)

The solution to this issue would be to "modify" the User/Portal bridging, replacing the UserPortals table by 3 new tables.

PortalGroups

This table would provide the new PortalGroups Entity

- PortalGroupID
- PortalGroupName
- PortalGroupDescription
- CanDelete

PortalGroupPortals

This table would provide the many-to-many relationship between Portals and PortalGroups.

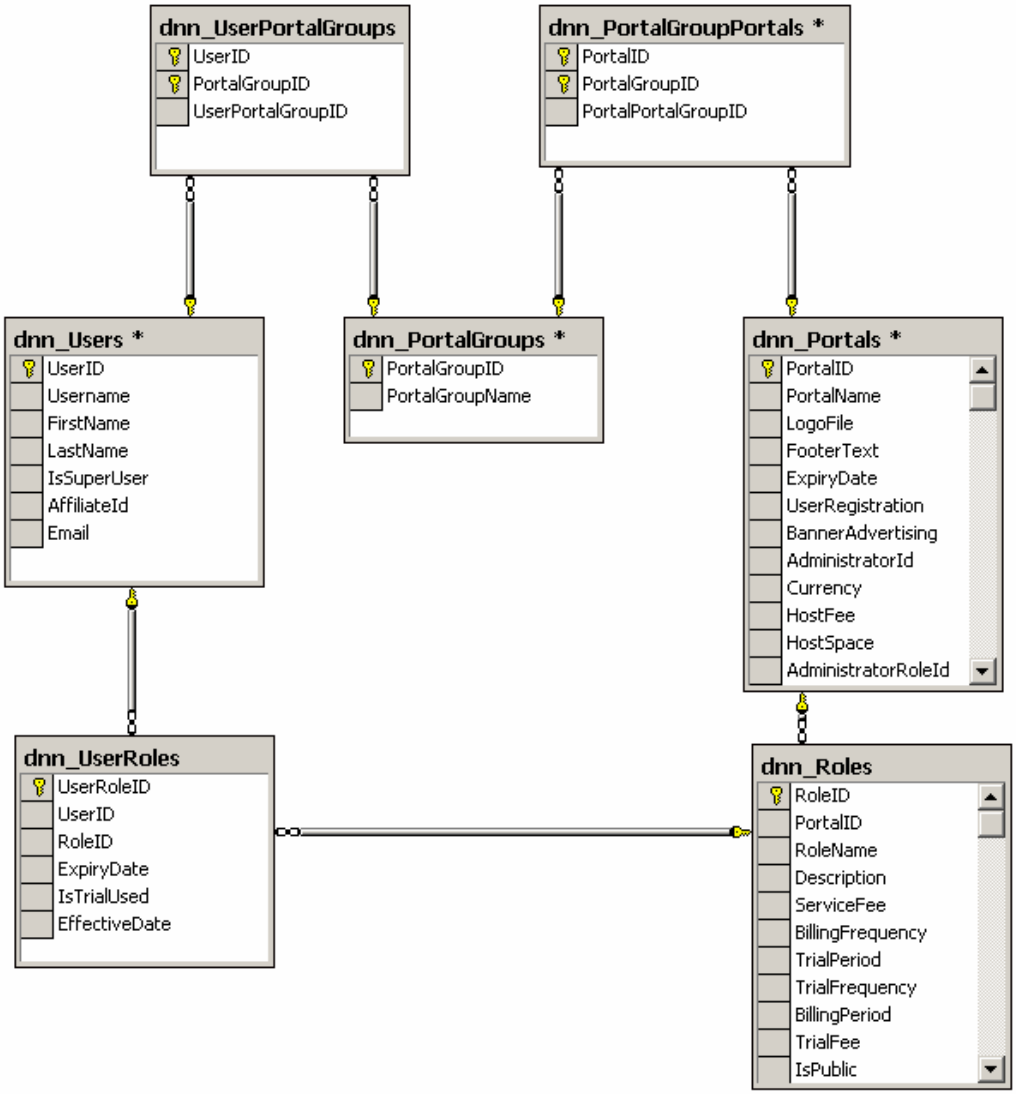
- PortalID
- PortalGroupID

UserPortalGroups

This table would provide the many-to-many relationship between Users and PortalGroups.

- UserID

- PortalGroupID



This arrangement provides a complete solution for option (3) - a User can be a member of any Portal Group and a Portal can be a member of any Portal Group. It would, however, require two types of "dummy" PortalGroups to satisfy the other two options – (1) and (2).

Option (1), one user / one portal - would require a dummy portal group to be created for each portal. Option (2), one user / all portals would require a dummy portal group that automatically contains "all" portals.

In reality, if we actually look at the 3 options, (1), and (2) are really the edge cases for option (3).

Membership Provider issues

Since now a user is a member of a Portal Group and not a member of a portal, we have some MembershipProvider issues.

ApplicationName

While there is a potential enhancement to "move" away from the use of HttpContext in cusomising the ApplicationName, should it be determined that this should not be done, then we need to be able to modify the ApplicationName to be derived from the PortalGroupId rather than the PortalId, as a username is unique for a PortalGroup rather than a Portal.

New Business Layer Classes

This model would require the creation of new classes to handle the new Database entities proposed.

PortalGroupInfo

- PortalGroupID
- PortalGroupName
- PortalGroupDescription
- CanDelete

PortalGroupPortalInfo

- PortalID
- PortalGroupID

PortalGroupUserInfo

- PortalGroupID

- UserID

PortalGroupController

This class would provide controller methods for all PortalGroup related functionality. The following is a list of methods that would need to be included – there may be others that are required.

- AddPortalGroup(...)
- AddPortalToGroup(ByVal portalId as Integer, ByVal portalGroupId as Integer)
- AddUserToGroup(ByVal userId as Integer, ByVal portalGroupId as Integer)
- DeletePortalGroup(ByVal PortalGroupId)
- GetPortalGroup(ByVal PortalGroupId)
- GetPortalGroupsByPortal(ByVal PortalId)
- GetPortalGroupsByUser(ByVal UserId)
- GetPortalGroups()
- RemovePortalFromGroup(ByVal portalId as Integer, ByVal portalGroupId as Integer)
- RemoveUserFromGroup(ByVal userId as Integer, ByVal portalGroupId as Integer)
- UpdatePortalGroup(...)

UserController issues

We also have a significant impact in UserController. Many methods currently "rely" on PortalId. Some of these would need to be changed to use PortalGroupId, although it may be preferable to "keep" these methods and just modify the implementation.

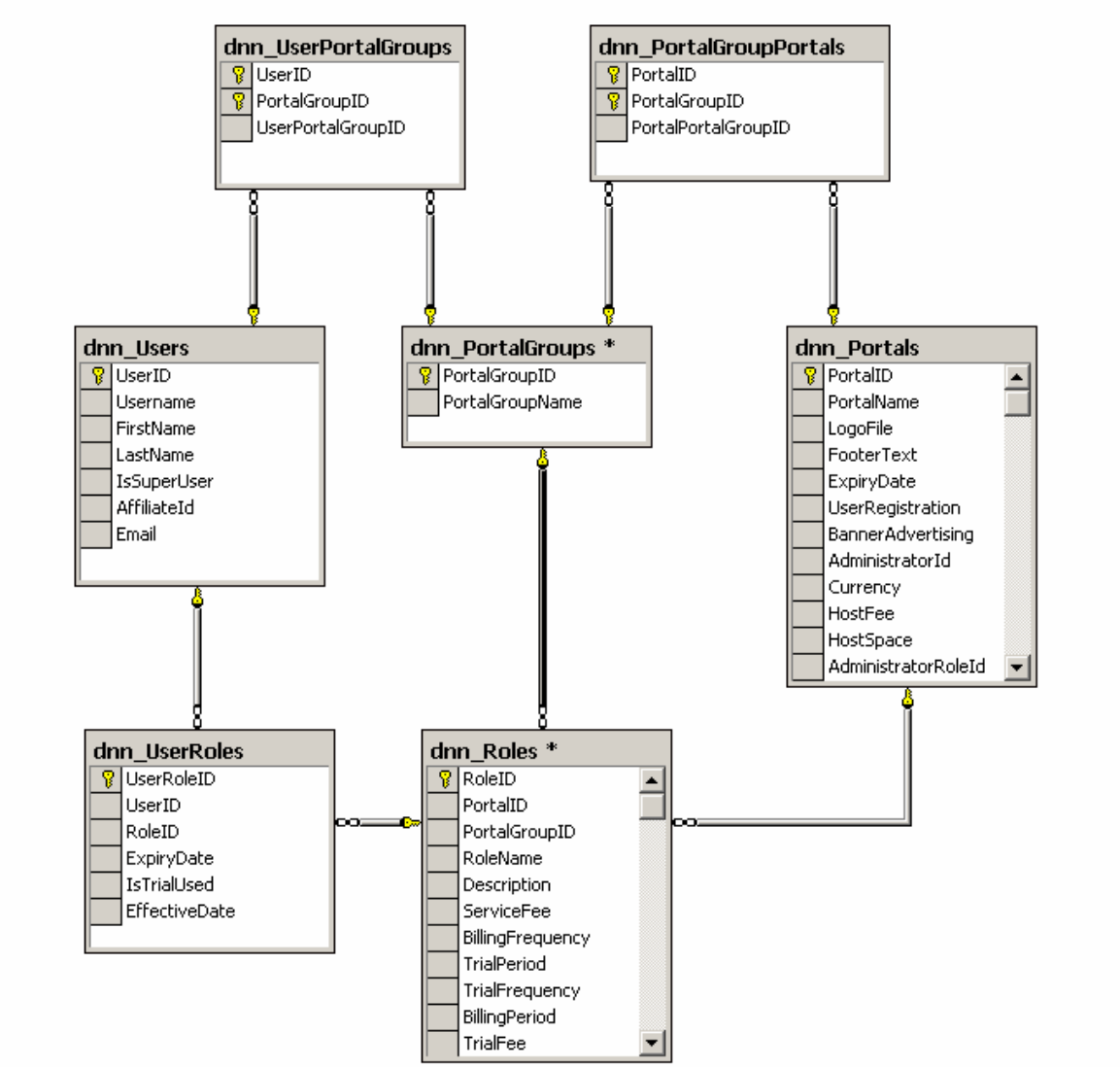
UI Changes

Depending on what changes are made in the Business Layer the impact to the UI, could be as little as adding the capability to manage the Portal/PortalGroup relationship. From the user side, a user registers with the Portal. If that portal is in one or more Portal Groups then the user is "registered" with those Portal Groups automatically.

Adding PortalGroup based Roles

One extension of the proposed enhancement is that Portal Groups should have their own roles “Group Roles”. These roles would then be “global” to the Portal Group, and membership in the Portal Group Role would carry across all portals in the Group. This behaviour is akin to the concept of Domain Users and Domain Admins in Active Directory.

This extension could be modeled in the Data Store as follows:



The change here is that in addition to the Roles table having a foreign key relationship with Portals (PortalID), it now also has a Foreign Key relationship with PortalGroups (PortalGroupID). This provides the "concept" that a Role is either a Portal Role (PortalID > -1, PortalGroupID = -1) or a Group Role (PortalID= -1, PortalGroupID > -1).

This modification would have the smallest impact on the system as a whole, as it would leave permission assignments to work the same as they do today. However, this idea still needs a significant amount of analysis before we are in a position to implement. I do believe that if we were to implement Portal Groups as described in this document we would be able to implement Group Roles at a later stage. It does not appear that there are any roadblocks being added.

Login to Portal Groups

Another extension to this enhancement is the "ability" to remain logged in when switching between portals in the same portal group. This ability would rely on the "cookie" used for forms authentication to be available to the Application.

Child portals that are part of a "portal group", with a url of www.mydomain.com/child will probably work this way as the cookie would still be considered as part of the same domain as www.domain.com. However, domains with completely different urls are a different matter as it is a potential security issue. A portal cannot "access" cookies that are not part of its "domain".

ASP.NET 2 introduces the "concept" of cross-Application Authentication. This is the ability to redirect to, and remain logged in, when transferring between Applications. We use a single Application, so this ability will not help.

However, ASP.NET 2 also adds the ability to set the domain to use for cookies in an application. In .NET 1.1 the domain is by definition the domain used in the request. A path value could be added, but the domain could not be set. In .NET 2.0, a cookie domain can be set in web.config. This can be set in one of two places. If the attribute is set in the <httpCookies> element it applies to all cookies, while if it is set in the <forms> element it only applies to the authentication cookie. So in theory this would allow us to provide the ability to retain the user's authentication ticket across portals in a portal group.

The cookie domain is set in web.config, so this would mean that the cookie will be available, in principle, to any portal configured on the system. This would then allow the

DNNMemberhsip module to determine if the “cookie” is valid for the current portal/group and act accordingly.

Summary

Risk Level: High – big changes to the core framework

Benefit: High – ability to market to Enterprise customers

Resources: 100 – 150 hrs

Removing Dependence on Context

Introduction

Currently the implementation of the AspNetProvider has a dependence on the HttpContext. The HttpContext is used to "store" the ApplicationName, which is used in the AspNetProvider's data store to partition the users. The problem with this scenario is that this means that is difficult to use this provider in order to access Membership information from outside the DotNetNuke application.

What needs to be done?

The dependence on HttpContext is due to our use of an overridden implementation of the SqlMembershipProvider etc. components of MemberRole. Each of the three classes implements a single overridden property ApplicationName.

```
Public Overrides Property ApplicationName() As String
   Get
      If Convert.ToString(HttpContext.Current.Items("ApplicationName")) = "" Then
         Return "/"
      Else
         Return Convert.ToString(HttpContext.Current.Items("ApplicationName"))
      End If
   End Get
   Set(ByVal Value As String)
      HttpContext.Current.Items("ApplicationName") = Value
   End Set
End Property
```

In simple terms, we can resolve our dependency issue simple by removing these overridden classes, and using the classes provided in MemberRole itself. In addition, everywhere in our AspNetProvider that we call SetApplicationName(portalId) can be removed.

For completeness, we should set the ApplicationName attribute in the web.config file to "dnn" or something similar, to partition DotNetNuke users from other (eg. Community Server) users.

That is as simple as the change would need to be for new installations. The big challenge would be for existing installations. Here we have two options:

1. Only provide an option to remove the HttpContext - existing installations can continue to work with the ApplicationName as it is managed to today, while new installations would use the new approach.

2. Provide a migration path. This would provide challenges for situations where there are two different users with the same username in two different portals. Although these users must be the same user (due to the restrictions imposed on the dnn_Users table), the profiles (including password) may be different in each portal, and would need to be merged.

What is the effect of this change?

While the change is simple enough, there is an impact on the "usernames" available to portals. This enhancement would mean that unless other mechanisms were implemented, there would be a restriction that once a username was used in a Portal then it could not be used for another portal on the same installation (unless the user was also a member of that portal).

Thus, this would work well with the Single-Sign/Portal Group options (2) and (3) discussed in the previous section, but would not work well for scenario (1), where the portals are not grouped or related in any way.

We must however remember that we are dealing here with a specific concrete implementation of an abstract provider. Not all concrete implementations are going to be able to completely support the abstract contract well. In this scenario, the AspNetProvider supports the contract but adds a restriction due to how it has to implement the contact.

DotNetNuke users who want to use scenario (1) from the previous section AND not have any dependency on HttpContext would need to use a different concrete implementation.

This is similar to the behaviour of the ActiveDirectory Providers provided by Microsoft in the .NET 2 Framework. These providers provide read-only access to the Active Directory Users/Roles (Groups) and cannot be used to modify the data store.

Summary

Risk Level: High – significant Upgrade issues

Benefit: High

Resources: 50 – 60 hrs

Password Enhancements

Introduction

There are a number of password enhancements that have been proposed.

- Offer Hashed passwords as an option
- Password reset by Administrator
- Implement Question and Answer for password retrieval or reset
- Enforce Password Complexity (min size, min alphanumeric)
- Password Aging/Expiry
- Force Password Update (or Profile update)
- Enable/Disable Password Retrieval
- Enable/Disable Password Reset

Most of these options are fairly simple to implement within the current framework, and could be added quite easily.

The only real issue is where we configure the option. The MemberRole component uses the web.config to configure these options, and we would need to configure them there for use with the AspNetProvider. However, we would also need to provide a mechanism for other concrete providers to provide these options.

The best approach would be to provide readonly properties in the abstract provider that match to the various options.

In the AspNetProvider concrete implementation the property would get the value of the attribute from the web.config attribute. Likewise any alternative provider would return a value, either from its own attribute collection, or a fixed predetermined value.

The benefits of this approach are that there is a convenient way for the API to determine whether the feature is supported or enabled. The drawbacks of this approach are that these features are then determined at the host level.

Hashed Passwords/ Password Reset

Hashed passwords would allow us to drop our dependency on machine keys (although this should be offered as an option rather than a change). For reduced support issues in the future the default could be changed to Hashed.

As hashed passwords involve a one-way encryption, we would no longer be able to retrieve the current password. This would need us to implement the concept of Password Reset (which is a related enhancement)

The current AspNetProvider supports the concept of Hashed passwords and provides a reset password option which will need to be used in this case.

New Provider Properties

PasswordFormat – an enumerated value PasswordFormat.Hashed, PasswordFormat.Encrypted etc.

New Provider Methods

ResetPassword – This method would need to be added so an Administrator (or a user who forgot their password) could trigger a new password to be automatically generated.

```
Public MustOverride Function ResetPassword(ByVal user As UserInfo, ByVal passwordAnswer
As String) As String
```

Changes to UserController Methods

ChangePassword – This method should work as before, as the current and new password need to be provided by the user.

GetPassword – This method would cause an exception to be thrown by the underlying memberRole component, which should probably be passed up by the AspNetProvider and caught by the UserController method.

ResetPassword – This method would need to be added so an Administrator (or a user who forgot their password) could trigger a new password to be automatically generated.

UI Changes

There are two changes that would need to be made to the UI.

SignIn.ascx – the SignIn page would need to determine the type of password being used and either call GetPassword (encrypted) or ResetPassword (hashed) before emailing the password to the User.

ManageUsers.ascx – the ResetPassword section, visible to an Administrator, would need to determine the type of password being used and either display the current UI (encrypted) or display a "Generate Random Password" button (hashed) tied to the ResetPassword API method.

Implement Question and Answer

Again the MemberRole provider supports this capability, so it should be fairly easy to implement.

New Provider Properties

RequireQuestionAndAnswer – a Boolean value that would determine whether the provider requires/supports this capability.

New Provider Methods

ChangePasswordQuestionAndAnswer – This method would need to be added so the user can set their Password Question and Answer.

GetPassword – An overload to this method would need to be added to get the Password. This overload would need to pass the passwordAnswer as a parameter.

ResetPassword – An overload to this method would need to be added to reset the Password. This overload would need to pass the passwordAnswer as a parameter.

Changes to UserController Methods

ChangePasswordQuestionAndAnswer – This method would need to be added so the user can set their Password Question and Answer.

GetPassword – An overload to this method would need to be added to get the Password. This overload would need to pass the passwordAnswer as a parameter.

ResetPassword – An overload to this method would need to be added to reset the Password. This overload would need to pass the passwordAnswer as a parameter.

UI Changes

There are two changes that would need to be made to the UI.

SignIn.ascx – the SignIn page would need to determine if the feature is enabled and if enabled provide the Password Question and an Answer TextBox when retrieving the Password.

Register.ascx – this page would need to provide the ability for a user to update the Question and Answer.

Password Complexity

Currently DotNetNuke enforces minimal password requirements. The MemberRole component provides three attributes (**minRequiredPasswordLength, minRequiredNonalphanumericCharacters** and **PasswordStrengthRegularExpression**). These could (and should be enforced at the API level).

New Provider Properties

MinPasswordLength – an integer representing the minimum password length.

MinNonAlphanumericCharacters – a number representing the mimimum number of nonalphanumeric characters.

PasswordStrengthRegularExpression – a regular expression that defines the password.

By setting API properties (that expose the value in the provider), the Business Layer can evaluate the password complexity provided and send a more meaningful message, back to the user – before attempting to update the password in the data store

Changes to UserController Methods

User controller methods that update the password would need to check the Provider Properties and then validate the provided password. This should probably be centralized in a new UserController method ValidatePassword, that the UI could then call during Page Validation to set/update a ValidationControl in the UI.

Password Ageing/Expiry/Update

There are a number of enhancements that can be grouped under the general concept of password ageing. While most of the other enhancements discussed are implemented in the MemberRole components, these components do not have any capability to deal with password aging or expiry, so they would need to be tacked on.

This property could be set at the Portal or Host level, but is probably best set at the Host level through a HostSetting.

Changes to HostSettings

Add a new HostSetting – PasswordExpiry – that takes an integer value. This would actually only need to be added in the UI, as the HostSettings API can handle additional settings.

Add an addition PasswordExpiryReminder setting that takes an integer value. This value would be used to trigger" a reminder to the user to change their password if within a certain number of days of the Expiry.

Changes to UserController

When logging in the UserSignIn method of UserController will log the user in if a valid username and password are provided. This method should be modified to check the value of the User.Membership.LastPasswordChangeDate property and compare it with the PasswordExpiry setting.

If the password has expired then the User should be rejected.

Changes to SignIn.ascx.vb

SignIn.ascx.vb will need to be modified so that if the user is rejected because of an expired password then the retrieve password button will generate an automatic password that will be sent to the user's email address.

It should also be modified so that after a successfule SignIn a "warning" as a ModuleMessage will be displayed if the user is within PasswordExpiryReminder days of their Password expiring.

Force Password/Profile Update

These enhancements are related. If an administrator has modified the profile properties, they may want all users to "update" their profile, especially if the new property is a required property (or they have changed an existing property to be required).

Similarly, if an administrator suspects that the User credentials have been compromised in any way, he/she may wish to force all Users or a single User to update their password.

Changes to User

Two new properties will need to be added to User – UpdatePassword and UpdateProfile. These property would then be used by the UI to determine if the current User needs to update their password and/or profile.

Changes to Users Table

Similar bit type fields should be added to the Users table in the Database to store the UpdatePassword and UpdateProfile flags.

Changes to MembershipProvider

The concrete membership providers will need to retrieve this new information from the Data Store and populate the User.UpdatePassword and User.UpdateProfile properties. In addition new provider methods should be added to set the flags for a single User or for all Users in a portal.

```
Public MustOverride Sub ForcePasswordUpdate(ByVal portalId As Integer, ByVal userId As
Integer)

Public MustOverride Sub ForcePasswordUpdate(ByVal portalId As Integer)

Public MustOverride Sub ForceProfileUpdate(ByVal portalId As Integer, ByVal userId As
Integer)

Public MustOverride Sub ForceProfileUpdate(ByVal portalId As Integer)
```

These methods would be used to set the bits, rather than the Add/Update user methods, and the Add/Update user methods would be modified to always switch off the bits.

Changes to UI

ManageUsers.ascx.vb – The User Administration UI would need to be updated to provide a mechanism for the Administrator to set the UpdatePassword and UpdateProfile flags for a User (or all users in the portal).

SignIn.ascx.vb – The SignIn UI would need to be modified to redirect the user to the Register page, in the event that the user's UpdatePassword/UpdateProfile flag is set.

Summary

Risk Level: Low – each enhancement is independent with minimal impact on the core.

Benefit: Medium – enhanced product offering for Enterprise customers

Resources: 10 – 20 hrs

Registration/SignIn/Logoff Enhancements

Introduction

There are a number of SignIn/Logout enhancements that have been proposed.

- CAPTCHA
- RedirectPage after SignIn (when from Home page)
- Email administrator after user lockout
- Redirect after logout (not always to Home)
- UrlReferrer needs to "retain" querystring params

also a number of Registration Enhancments have been proposed.

- CAPTCHA
- Public Registration – send email to user in case the email used is not the user registering
- Send verification url that links directly to the "SignIn page"
- Send email on profile change to alert user in case an Unauthorized user has updated their profile.

Most of these enhancements have little or no impact on the API. The first four are features that should be set through the use of PortalSettings.

New PortalSettings

In order to support these enhancements we could add the following new PortalSettings

- UseCaptcha (could have two options here – use for SignIn, use for registration)
- Email Administrator after user lockout
- SignIn Redirect Page (if logging in from Home)
- Logoff Redirect Page

Changes to UI

SignIn.ascx.vb – The SignIn module would need to be modified to either redirect the User to the referring page, or if that page is the HomePage to redirect the user to the SignIn Redirect Page. In addition the Urlreferrer would need to be parsed and the querystring parameters saved so that the user can be redirected to the appropriate sub-item.

Add support when the control loads to read the validation code from the "url", rather than requiring an additional step.

Logoff.aspx.vb – Similarly the logout page would need to check the Logout Redirect Page setting rather than automatically redirecting to the Home page.

Regsiter.aspx.vb – Send an email to the user on registration, even in Public registration mode, so the person with the email knows it is being used to register on the site. In addition add a direct "url" link to the login page, with the validation code included so the user can login immediately.

Also send an email when the profile is updated, so the user is aware if an unauthorized user is trying to change their profile.

Summary

Risk Level: Low – each enhancement is independent with minimal impact on the core.

Benefit: Medium – enhanced product offering for Enterprise customers

Resources: 10 – 20 hrs

Integration of Users Online API

Introduction

Our Users Online capability is currently separate from our Membership components, even though our current provider AspNetMembership does support the concept of both Anonymous users and isOnLine, so could be used to provide the Users Online information. This enhancement proposes that we integrate the UsersOnline API with the other Membership components.

Current Users OnLine Architecture

UsersOnline HttpModule

The current architecture uses the UsersOnline Http Module to track whether users are online. If the HostSetting is enabled on every AuthorizeRequest event the UsersOnline module calls the TrackUsers method of the UsersOnline Controller.

The system uses two settings to determine whether a user is online:

- DisableUsersOnline – is Users Online enabled/disable
- UsersOnlineTime - the window in minutes (ie has there been any activity within the last x minutes) defaults to 15 mins.

If the user is authenticated (ie a User in the system) the TrackUsers method calls TrackAuthenticatedUsers. If the user is not authenticated AND there are cookies in the request, TrackUsers calls TrackAnonymousUsers.

TrackAuthenticatedUsers updates (adds if necessary) the cached list of Online Users, creating a lightweight user object for the authenticated user and saving it to the Cache.

TrackAnonymousUsers updates (adds if necessary) the cached list of Online Users by creating a lightweight anonymous user object with a temporary user Id and saving it to the cache. In addition the temporary id is persisted to a cookie in order to track the anonymous user.

This process updates the user cache on all AuthorizeRequest events, but as we will see below the data is only persisted to the data store when the PurgeUsersOnline Task is run.

PurgeUsersOnline Scheduled Task

As the Users Online Information is "cached" in memory, periodically the information is purged, by persisting the cache to the Database and clearing the cache. This is the extent of the core's involvement in Users Online.

Users Online Desktop Module

The users online desktop module displays the current Users Online information. This module actually "bypasses" the standard DotNetNuke practice and directly calls data layer methods – GetStatistics and GetOnlineUsers.

This module should probably, either call core UsersOnline methods for this information OR have its own business controller class that it can call to access the database information.

The net result however is that this module fetches its information from the DataBase (bypassing the cached list), and therefore relies on the frequency of the PurgeUsesOnline scheduled task.

MemberRole Users Online Capabilities

The MemberRole components have some similar online users capabilities, although the process it uses is slightly different.

Authenticated Users

For authenticated users, MemberRole updates the LastActivityDate field of the aspnet_Users Table – when the following methods are called.

- CreateUser – updates the table on successful creation of a new user

- GetUser – optionally will update when a user is retrieved.
- ValidateUser – updates the table
- UpdateUser – updates the table with the LastActivityDate as defined in the MembershipUser object.

Our current implementation sets the LastActivityDate to Now whenever the user is updated in the database. The underlying MemberRole components also automatically update this value if the Profile is changed (even if changed by the Administrator).

If we were to use the Membership Provider to manage Users Online we would need to modify that to only be "updated" if the updates were triggered by the user him/herself.

So if we were to "fix" our implementation of MemberRole the DataStore would reflect the Last Activity based on "Login", "UserUpdate" and "ProfileUpdate", rather than every page request..

Anonymous Users

The MemberRole does provide the capability to "track" anonymous users using a cookie and the ability to track the user's profile, but it does not provide any functionality to track their latest activity.

Integration of UsersOnline

It is clear from the above analysis that there are limitations to the MemberRole's online activity functionality. Having said that, there are also deficiencies to our current implementation, and there is no reason why the current functionality should not be "moved" to the MembershipProvider, so that all Membership/User services are in one place.

UserOnlineController

In order to provide core support for User Online functionality the following functionality should be added.

- GetOnlineUsers – gets a list of users that are currently online. This method is likely of interest to other modules, such as forums so it is definitely a core service.

- GetMembershipStatistics – returns the statistics used by the Users Online module. This method could be left in the UsersOnline Module, as it is of specific interest to that module. If it is not added to the core as a service the Users Online module should be updated to include a Controller class that calls the data layer, so as to support DNN Best Practices.

In theory this class could be deprecated and all the UserOnline functionality "moved" to UserController.

Additions to MembershipProvider

The following methods currently in the core DataProvider or in the UsersOnline Module's DataProvider should be moved to the Membership Provider and associated SqlDataProvider.

- GetOnlineUsers
- DeleteUsersOnline
- UpdateUsersOnline

These methods would then be called by the UserOnlineController class, instead of the existing "DataProvider" class. This leaves the implementation of Online status to the Membership Provider.

Other Enhancements

IsOnline - The UserMembership object has an IsOnline property that is currently populated from the AspNet MembershipUser object. It would be "useful" to actually populate this value correctly (see earlier discussion). This could be done by modifying the GetUser/GetUsers methods (and related stored procedures) to return the value. Thus the Users UI could indicate to the Administrator whether the user is currently online.

Summary

Risk Level: Low – for the most part this is just an extension of the refactoring already carried out in the abstraction effort

Benefit: Low – the benefits are primarily in a simpler API

Resources: 10 – 20 hrs

Demo User Accounts

Introduction

This enhancement proposes the concept of a user account that provides limited ability to update the account. An administrator could provide a single "Demo" account that allows the user limited access to some areas of the site.

As a mimimum, the demo user should not be able to modify the password (this can only be done by the administrator).

A further extension could be that demo accounts are not automatically considered to be members of the Registered Users role, but have their own role Demo Users.

Implementation

The easiest way to implement this functionality is to add an IsDemo property to the UserInfo class (and the equivalent IsDemo column in the Users table).

In addition the ManageUsers.ascx UI (but not Register.ascx) would include an IsDemo checkbox. This checkbox would be used to set the IsDemo property.

Finally, the Register.ascx UI would need to be modified to only display the Update Password section if the IsDemo property is false.

The extension to automatically add a Demo User to a Demo Users role (and not to the Registered Users role) would require two things to happen.

1. The Demo Users role would have to be "created" on each new Portal Creation and configured as IsPublic = false and AutoAssign=false.

2. In UserController.CreateUser the code to autoassign roles would need to be modified to:

a. Automatically assign a User with IsDemo=true to the Demo Users role, and

b. Suppress assigning a User with IsDemo = true to any other Autoassignment role.

Summary

Risk Level: Low – enhancement is additive and has little or no impact on current functionality

Benefit: Medium – enhanced product offering for Enterprise customers

Resources: 10 – 20 hrs

Approved Status Enhancement

Introduction

This enehancement proposes to modify the “IsApproved” status, so that it except multiple values (as an enumerated type) rather than the single true/false. This would allow the ability use Approved/Banned and other statuses as necessary in the future.

Implementation

While a complete implementation of this capability would need an understanding of the types of status needed, he implementation of this is fairly straightforward from an API perspective.

UserMembership Class

The type of the Approved property in the UserMembership class would need to be modified to the new enumerated type. It should also probably be renamed Status.

Impact of the change in other areas.

There are 21 places through the Application where this property is either read or set. These would need to modified to set the property to the new correct enumerated value, rather than True or False.

In addition the field would have to be added to the dnn_Users table as the AspNetProvider only supports a simple true/false setting, and the appropriate stored procedures that Add/Get/Update this table would need to be modified accordingly.

Once these changes were carried out then the “framework” would be set up for the use of other statuses with other meanings.

Summary

Risk Level: Low –enhancement is fairly straightforward and is little more than a refactoring exercise.

Benefit: Medium – enhanced product offering for Enterprise customers

Resources: 10 – 20 hrs

Additional Information

The DotNetNuke Portal Application Framework is constantly being revised and improved. To ensure that you have the most recent version of the software and this document, please visit the DotNetNuke website at: http://www.dotnetnuke.com

The following additional websites provide helpful information about technologies and concepts related to DotNetNuke:

DotNetNuke Community Forums
http://www.dotnetnuke.com/tabid/795/Default.aspx

Microsoft® ASP.Net
http://www.asp.net

Open Source
http://www.opensource.org/

W3C Cascading Style Sheets, level 1
http://www.w3.org/TR/CSS1

Errors and Omissions

If you discover any errors or omissions in this document, please email marketing@dotnetnuke.com. Please provide the title of the document, the page number of the error and the corrected content along with any additional information that will help us in correcting the error.

Appendix A: Document History

Version	Last Update	Author(s)	Changes
1.0.0	Feb 3, 2006	Charles Nurse	• First Draft
1.0.1	Feb 7, 2006	Charles Nurse	• Second Draft