Best Practices for DNN Extension Development

Published July 2020

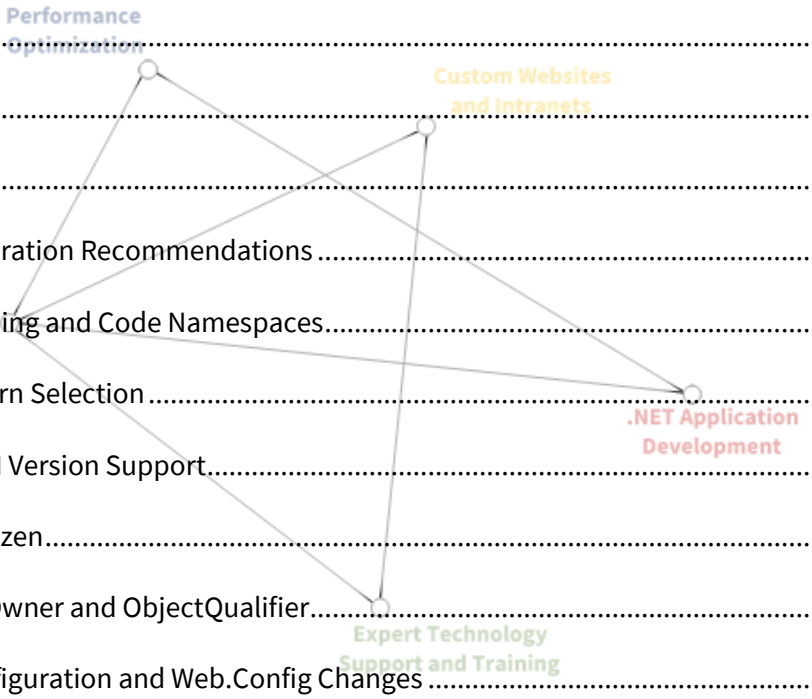
IowaComputer GURUS

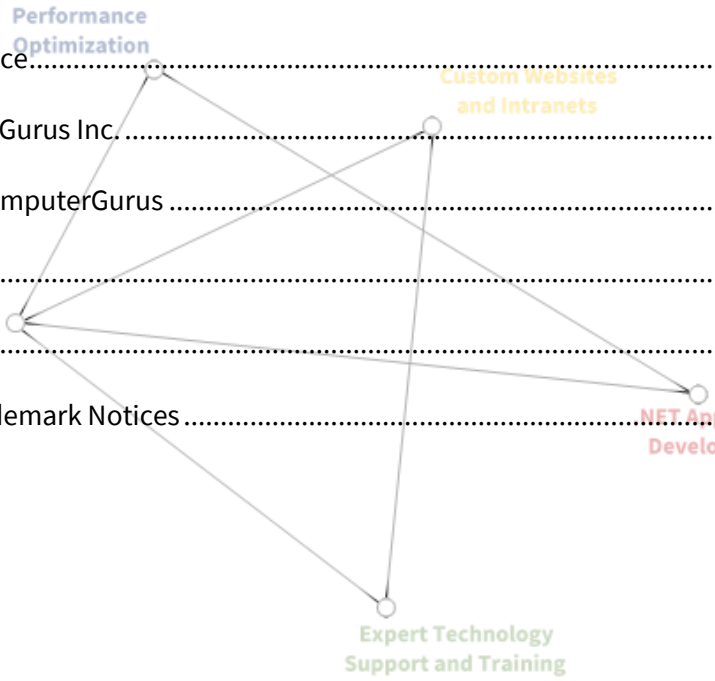
Design **Develop** **Optimize** **Support**

Technology Services and Support . . . for the Life of Your Project

IowaComputer GURUS
515.270.7063 | support@iowacomputergurus.com

Contents





IowaComputer GURUS
515.270.7063 | support@iowacomputergurus.com

The Purpose of This Document

This document has been created to provide a baseline set of information related to the development of Extensions for the DNN / DotNetNuke Platform.[1] The best practices listed here cover items that relate specifically to the creation of Modules, Authentication Providers, and Skin Objects and have been compiled based on information collected from the DNN Community and over more than 1,000 custom development projects that have been completed by IowaComputerGurus over the past 12 years.

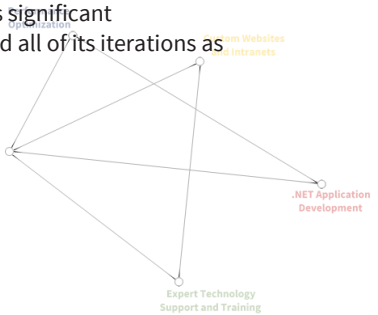
Not all aspects of each section will apply to every project. Also, even with the best of intentions, some project solutions must deviate from the practices listed in this guide to be effective. Therefore, we want to stress that this is simply a set of basic best practices that should be applied in most situations whenever possible, to ensure performance and interoperability.

Who Should Use This Document?

This document is NOT intended to be an introductory course on how to develop DNN/DotNetNuke modules. However, where applicable, we have included additional references to community-based and other resources that might be of benefit for those just starting with DNN/DotNetNuke development.

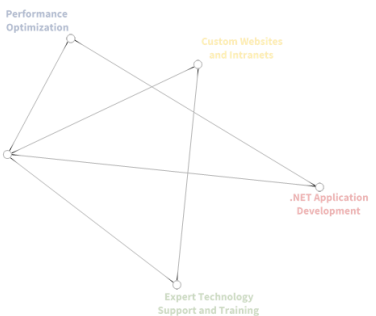
This document is intended to be used by developers with at least an intermediate understanding of general development principles. Specific knowledge of DNN is not required, but a basic understanding of software and website development is assumed.

[1] DNN Software—the makers of the platform—have changed the name "DotNetNuke" to DNN, and created additional product tiers under the brands such as "DNN EVOQ," etc. Generally speaking, the terms DNN, DotNetNuke, and DNN Platform refer to the open source "free" versions of the product. The terms DNN Professional, Evoq, Evoq Content, Evoq Content Basic, DNN Social, Evoq OnDemand, and Evoq Engage are premium or paid-for versions. Iowa Computer Gurus has significant experience on all versions. For the purposes of this document we will hereafter refer to the platform and all of its iterations as "DNN" unless there is a specific reason to do otherwise, such as code or API related phrasing.



Document Revision History

Rev. Date	Notes
2/7/2009	First public release of module.
7/3/2012	Updated for DNN 6.x support and design patterns.
5/15/2015	Updated for new document format.
10/01/2015	Format and content changes, updated links, external references, and version notes.
7/10/2020	Updated to reflect current issues impacting the DNN community and edited for relevance since the prior revision.



Getting Started with Module Extension Development

As previously stated, this guide was not created to provide an introduction to DNN module development, but to be a guide in creating modules that have the best chance of being stable across the multitude of DNN environments. The following sampling of links and resources may be helpful in "Getting Started" with DNN development.

Project Templates

In recent years the number of templates available to developers has changed, with one only one of the most commonly supported ones remaining in active development. These serve as a simple starting point for development, however, should be used with caution.

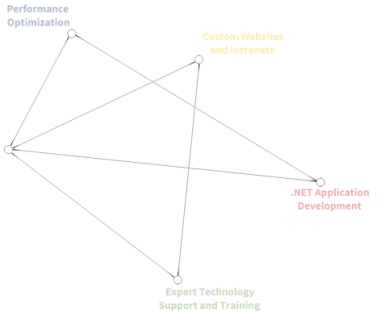
Chris Hammond's Template

Chris Hammond has been managed development templates for DNN for the better part of a decade. His templates can be obtained from his project GitHub page https://github.com/ChrisHammond/DNNTemplates. These templates provide both VB.NET and C# templates to help get started with DNN Development.

As with any template, these contain multiple bits of boilerplate code, and it is essential to review these templates and any generated projects and REMOVE the unused boilerplate content. Failure to do so can result in errors and other unexpected behaviors. IowaComputerGurus strongly suggests adapting your templates that are tailored to your purposes.

Online Resources

There is a plethora of information available online regarding DNN development. However, due to the massive amounts of information available, it can be hard to find the exact information that you are looking for. Therefore, the following are a few of the recommended online resources for those looking for basic development information.



DnnCreative.com

DNNCreative is a paid video tutorial site; however, they provide an extensive collection of tutorials that can be helpful for the developer that is looking to start module development as well as individuals looking for basic DNN usage information.

DNNCommunity Blogs

The blogs on dnncommunity.org are one of the most commonly overlooked information sources available. The articles here are written by other community members showcasing the important lessons they have learned. You will also find updates from the project organizers about upcoming releases and more.

MitchelSellers.com

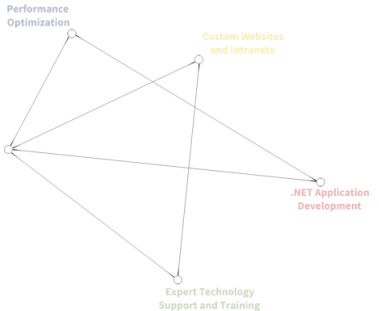
In addition to the above resources, our CEO Mitchel Sellers will often write short articles about various DNN and .NET development topics on his blog: http://mitchelsellers.com/

Books

There are many DNN books out on the market. There is only one book relating exclusively to DNN module development, "Professional DotNetNuke Module Development," written by Mitchel Sellers. Although this book covers a good foundation, it is worth noting that it was authored in 2009 around the release of version 5.0.0 of DNN Platform, where the current version is 9.6.2 as of July 2020. Although slightly outdated, it is still one of the few reference materials that cover development topics in detail.

Project Setup/Configuration Recommendations

This section supports the project setup, configuration, and packaging for deployment. These recommendations are general and apply to all types of extension development. The goal of these recommendations is to make your project easy to manage while limiting any possible risk of collision with other projects.



Module Folder Naming and Code Namespaces

Developers often overlook folder names and code namespaces during the project setup process, however, careful consideration should be taken when selecting these values due to the way DNN works. DNN users can install a module developed by any company, at any time. This flexibility can be a risk if using common naming for projects. As such, we have two recommendations for module folder naming: the Best Scenario, which provides for the best security against conflicts and the Best Compromise Scenario, which is slightly easier to implement, but can still have risks, although, significantly minimized. The final point in this section discusses the importance and recommendations of Namespace and Assembly definitions.

Best Scenario

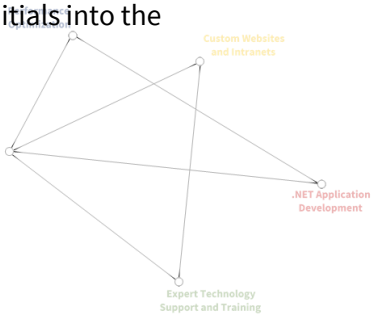
From a Best Practices standpoint, we have found that placing all custom modules developed by a single company into a shared folder is one way to prevent conflicts from a physical file point of view. For example, placing a "guestbook" module developed by IowaComputerGurus in the /DesktopModules/ICG/Guestbook folder would provide a level of separation between other modules that traditionally are stored directly inside the /DesktopModules folder and our module.

NOTE: This same concept can be used for MVC modules, which must live in the /DesktopModules/MVC folder by introducing a similar parent folder such as /DesktopModules/MVC/ICG/Guestbook.

This naming convention helps ensure that the only file conflicts that would be introduced are explicitly related to the code that you deploy. The process of configuring this is slightly involved, requiring the changing of the base path and other path values in the respective .dnn manifest files. The overall level of effort for new projects is less than 1 minute of configuration. Once deployed, it can be harder to adjust due to the install/update processes.

Best Compromise Scenario

If, for one reason or another, you cannot work with the "Best Scenario" laid out in the previous section, working with a naming strategy that incorporates your company name or initials into the



module folder is an excellent place to start. For example, with the guestbook placing the module in an ICG_Guestbook folder would reduce the risk of conflict. This scenario is not as favorable only because it does not group modules together.

Namespaces and Assembly Names

Aside from physical file naming within the module folder, namespaces and assemblies must be created in a manner that will not conflict with others as well. Using a namespace structure following the format Company.Modules.ModuleName or ICG.Modules.Guestbook for example helps ensure that others will not be placing code inside a similar namespace where it might conflict with your code.

The last consideration is in the naming of the compiled DLL. We recommend following Microsoft naming guidelines and using the root namespace for the project as the assembly name. In our example, we would generate a DLL with the name of ICG.Modules.Guestbook.dll. This method ensures that we have the best chance of not conflicting with another module. If we simply used Guestbook.dll it could be easy for another module to have the same DLL name and overwrite our file.

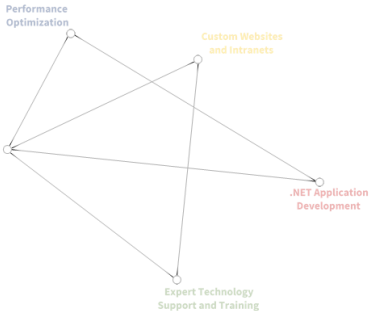
Development Pattern Selection

DNN platform currently supports numerous development patterns; WebForms, MVC & SPA as the primary patterns, each of these patterns are at different levels of maturity and support as it relates to the features and functionality that you can implement.

Overall, the decision point for most developers will focus on which toolset is best for their skillset. However, it is important to note specific situations that might dictate a model selection.

Authentication Providers

DNN supports the concept of Authentication Providers, whereby as an extension developer, you can inject a custom authentication process. This can be used to support two-factor-authentication or integrate with third-party systems.



When working with this project type, you MUST utilize an ASP.NET Web Forms User Control to get proper integration to the DNN Platform. Internally to this, you can do additional customization but the UI framework for login must be WebForms.

Integrations Requiring DNN's Rich Text Editor

DNN Platform has a built-in rich text editor, currently based on the CKEditor. The DNN implementation of this editor includes helpful features such as an integrated file browse/upload solution. This is implemented as a WebForms user control and can only be used by WebForms User Controls.

Themes & Skin Objects

Any developed theme for the DNN platform must be based on either a parsed HTML template or a Web Forms User Control. Additionally, any element to be injected into a Theme as a Skin (Theme) Object must be implemented with WebForms.

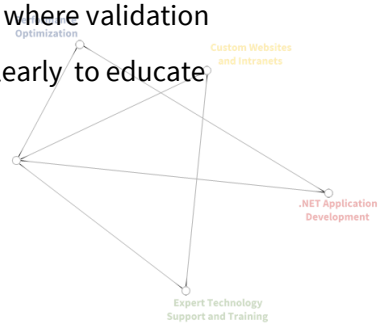
MVC Limitations

Although DNN Platform does provide support for ASP.NET MVC, it is important to understand that the MVC implementation is being executed within the context of an ASP.NET WebForms page lifecycle. This introduces certain limitations in functionality, API's and other behaviors. Most notably is that standard auto-implemented ASP.NET MVC Unobtrusive validation that many developers have come to expect DOES NOT work with DNN platform as of version 9.6.2.

Mixing Patterns

One common misconception with the various DNN Platform models is that you must "pick one," this is not a true statement. However, it is easier to pick just one. Nothing within the DNN Platform code prevents a developer from using a combination of WebForms, MVC, or SPA within the same extension solution aside from any previously mentioned "hard stop" requirements.

Mixing patterns can be a powerful tool for teams that which to utilize efficiencies of the Razor Syntax for MVC modules for UI intensive elements and fall back to WebForms for situations where validation is vital. IowaComputerGurus strongly recommends documenting these situations clearly to educate



development teams regarding the nature of any decisions as often the domain knowledge of why can be lost over time.

Packaging and DNN Version Support

DNN has evolved dramatically, and there are various limitations and risks for supporting specific versions. Each project will need to consider the audience they are supporting and any tradeoffs or limitations encountered by making a particular target decision.

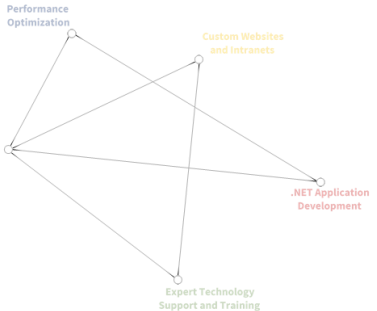
Select the Highest Possible Minimum Version

As a general rule targeting the highest possible minimum version of DNN for any custom development is the best possible starting point. This provides you with a number of benefits:

- Targeting the latest version, you can take advantage of the most currently added functionality. With recent changes in DNN, this gets you access to things such as Dependency Injection (Added in 9.4.0) as well as improvements in Search Integration (Added in 9.6.0) or similar
- Future API changes are documented during various releases along the lifecycle of the DNN Platform project. The project does follow a process for removal of items; however, if you retain too far back, you might not be aware of possible API removals that could impact your project in the future.
- .NET Version upgrades have occurred in recent releases, such as an update to 4.7.2 as a requirement for 9.4.0 and later.

For those that are developing solutions for wide-spread distribution, it might be necessary to target an older version to support a broader audience. However, IowaComputerGurus' recommendation for 2020 is to target nothing earlier than 9.2.0. With the number of security issues resolved and the severity of the problems addressed, we need to encourage those on older versions to upgrade, and going back too far limits the ability to build software with current development standards.

For those that must target 8.x, or even 7.x to support customers, it is recommended to update your project, even temporarily, to validate for any potential build warnings, or errors, after the release of each new DNN version; allowing for proper testing for upcoming breaking changes.



Always Set the Minimum Required Version

Once you have decided on which version to target, it is crucial to ensure that users do not install an extension on a non-supported version. If you build your module against DNN 5.5.0 and a user tries to install it to a DNN 5.0.0 website, as soon as the package is unzipped, their site will cease to function, throwing an error that it was not able to bind to the proper DLL. Rolling back from this situation requires developer experience or a rollback to a system backup, both of which cost time & money for website owners.

To ensure that this does not happen, all DNN manifests support the specification of a version limiter. If the DNN installation is not at a set minimum value, the module will not install, and the user will be notified that they do not yet meet the pre-requisites for installation. You set this version by adding the following nodes under the main <package> node within your DNN manifest and is included by default with most recent templates, including those from Chris Hammond

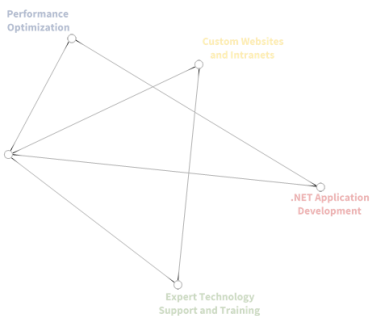
```
<dependencies>
<dependency type="CoreVersion">06.00.00</dependency>
</dependencies>
```

It is important to note that the CoreVersion value must be a three-part version number with two digits for each part, as reflected above.

Do not forget to update this value if/when you update your NuGet references for DNN in the future!

Being a Good DNN Citizen

In the preceding section, we reviewed the most critical elements for development to ensure that your solutions will work, continue to work with the evolution of the DNN Platform and some basic features. This section switches the focus a bit for those that are developing solutions for wider distribution, whereby following more of the DNN standard behaviors can become crucial, even if you might not use a feature yourself.



Support DatabaseOwner and ObjectQualifier

One key configuration element of DNN that is often overlooked by module developers is the ability to specify custom DatabaseOwner and ObjectQualifier values. Items not heavily used, and then being omitted from SQL Data Provider files can prove to be problematic and a support nightmare.

Supporting these replacement tokens, {databaseowner} and {objectqualifier}, is incredibly simple from a developer standpoint. We have found that when creating scripts via an external system, it is easy to do a find and replace operation to add this support as long as you follow a specific naming standard. For example, if you name all objects using [dbo].[objectname], you can do a find on the value "[dbo].["without the quotes and replace it with "{databaseOwner}[{objectQualifier}" again without the quotes, and your entire script set is updated.

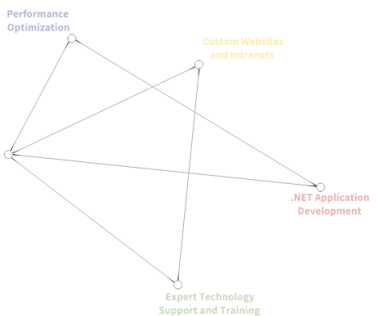
IowaComputerGurus does NOT condone or recommend utilizing these options; however, since they are included in the platform, anything distributed broadly should support the pattern.

Avoid External Configuration and Web.Config Changes

Although the XML Merge functionality it is easy for developers to make web.config changes without requiring site administrators to make manual changes; we recommend that solutions avoid any web.config changes if at all possible. By avoiding these configurations, you have a module that will install in a more streamlined approach, and there are fewer moving parts that can go wrong.

Even with the XML Merge functionality that comes with 5.x and later there are still risks associated with adding web.config items and conflicts with other modules. Although we fully understand that, depending on the specific implementation, this may not be possible. So this rule is something that should be considered carefully based upon what goal is being attempted with the custom solution.

If you must utilize a configuration merge to install your extension, pay special attention to the uninstall step to ensure that when removing your module, it will allow a clean rollback.



Avoid Manipulating DNN Database Tables Directly

When reviewing custom solutions that have been developed for customers in the past, one of the most common mistakes that we see are cases where existing DNN API's are not leveraged. Developers are either re-creating the wheel or are manipulating DNN database tables and data directly. To be clear, you should AVOID doing anything to DNN tables directly. We will help you understand how to compensate for this in the following sections. They will give some insight into where you can find the API's necessary to leverage the DNN functionality the reasons why direct database access is risky.

Determining Proper APIs

Sadly, one of the areas lacking within the DNN framework is API documentation. There is a bit of a learning curve necessary to fully identify the various namespaces and locations that can be used to obtain the information needed. The following table lists a few of the most common namespaces that provide access to key DNN data.

Namespace	Description
DotNetNuke.Entities.Portals	Classes in this namespace provide portal specific information. The PortalController is one of the key objects to investigate in this namespace.
DotNetNuke.Entities.Tabs	Classes in this namespace provide information on tabs, or pages, within the DNN portal. The TabController provides many helpful methods, especially for obtaining lists of pages in the portal, and similar items.
DotNetNuke.Entities.Modules	Classes in this namespace provide information on modules within the DNN portal. The ModuleController class provides access to module lists, settings and more. This is one of the more commonly known namespaces as the default "settings" functionality uses the ModuleController.
DotNetNuke.Entities.Users	Classes in this namespace provide information on users within the DNN portal. The UserController provides most

Namespace	Description
	of the basic functionality needed to retrieve and work with users.
DotNetNuke.Security.Roles	Classes in this namespace provide information about Roles. The RoleController allows you to get a list roles, add/remove users from roles and more.

The above reference serves as a good starting point and illustrates that DNN does expose a very robust API for programming. The only real limiting factor is that sometimes you may have to experiment with a method just to see what it does due to the lack of API comments.

Risks of Direct Database Manipulation

From a best practice standpoint, the real question here is, "what is my risk"? Well, we believe that the risks of direct database manipulation are very high, thus the inclusion of this topic in our best practices recommendation. Looking at risks, we have a few vectors to investigate.

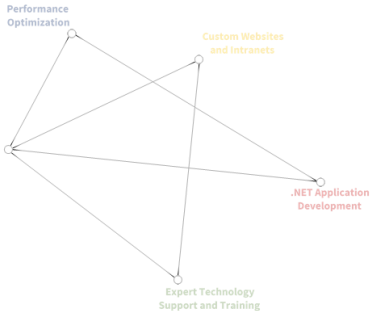
Upgrade Risks

The primary risk with direct database manipulation comes with future upgrades to DNN. No set criterion prevents DNN from changing the structure, format, or inputs for core tables and procedures. Therefore, if you are calling database assets directly, your custom scripts can either fail or result in unintended operation. This risk is tough to quantify.

Depending on the nature of your solution, it is also possible that your customization to the DNN tables could result in upgrade failures. Breaking changes are especially common when adding Indexes or Triggers against a core table.

Operational Risks

Aside from the upgrade risks, bypassing the API's can have operational risks or effects. For example, directly modifying information on modules that support caching, or user information that is cached by



DNN, will not update. If you use, the API's provided the respective cache entries are purged, and the information is updated.

roperly Tie Data for Multi-Portal and Multi-Module Installations

Another Best Practices area surrounds the whole concept of data isolation. What DNN data element should drive the storage of your data? Our Best Practices recommendation in this area are mostly focused on using common sense. The following table outlines the three most common data elements used:

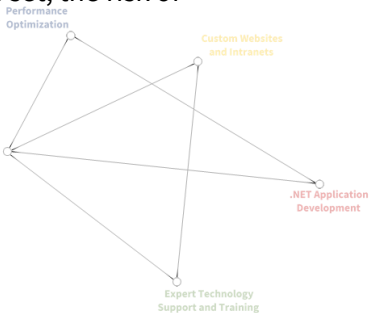
DNN/DotNetNuke Data Element	Use For Storing
ModuleId	Individual settings for a module and data for a module. This data, if a module is "referenced", will still be visible for the referenced module.
TabModuleId	Individual settings for a module, typically no data should be isolated at this level. If a module is put on multiple pages, it will have one ModuleId value and a different TabModuleId value for each page that has been added to.
PortalId	Use this for storing data at a portal level.

Selecting the proper data element is really a matter of understanding your requirements and how long the data should stick around. Therefore, our recommendation here, use appropriate options.

Foreign Keys and DNN Tables

One extension to this Best Practices is how to handle foreign keys when it comes to custom tables and their respective data relations. There are two schools of thought here, input the values in the custom tables but don't tie anything specifically to the DNN tables or, tie the tables using Foreign Key constraints. Be sure to set ON DELETE CASCADE.

It has been our experience that the latter is the true Best Practices as it helps keep old data from cluttering up a system. When a module is deleted, as long as ON DELETE CASCADE is set, the risk of



impacting a delete operation is minimal. However, if you forget this option, you can cause serious troubles within the DNN system.

Use Consistent Flow for Module Configuration

Many developers come up with various ways of configuring their modules. Some use the standard "Settings" control method in which you use the default "Settings" action menu item and a custom control with a key set to "Settings". Others add specific action item elements to the modules and require that users go through a separate process.

It has been our experience that whenever possible, a centralized management point is critical for proper module configuration and ease of use. For this Best Practices, we recommend using the standard Settings option. However, we fully understand that this is not the friendliest method if a number of settings are needed as screen real estate is at a premium. If another method of configuration is needed, be sure to add guidance to users, avoid requiring multiple settings forms to be filled out. If multiple configurations are needed, consider implementing a "Control Panel" style configuration to ensure ease of use.

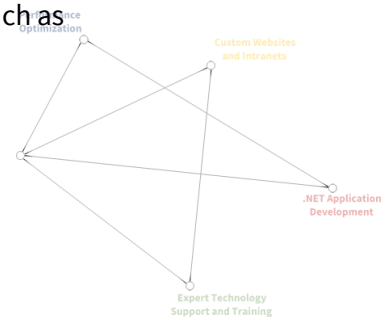
Carefully Consider any Third-Party Inclusions

One of the essential actions to consider when looking at being a good DNN Citizen is to carefully consider the inclusion of any third-party DLLs with your solution. The dynamically configurable nature of the DNN Platform is where you can encounter issues. Sure, you might want to utilize Entity Framework in your third-party module. However, another vendor may also. Given that this is NOT a dependency distributed by the DNN Platform, you could introduce a different version of the requirement that might break another module, or they could do the same to you.

Thankfully, the DNN platform does provide some necessary support to help avoid these conflicts.

Register ALL DLL's using an Assembly Node With Version

To have a DLL file placed in the /bin folder at installation you must have an <assembly> node within your DNN manifest. This node optionally supports a three-part version identifier such as <version>01.00.00</version.



Once specified, the assembly is registered to the system. If another extension is installed that attempts to register an older version, it will fail to install. If an extension is introduced with a later version DNN Platform will automatically add an Assembly Binding Redirect to the web.config and update the version of the assembly in the /bin folder.

The addition of a binding redirect provides necessary prevention for significant changes. It is still possible for major/minor version jumps to cause issues, but the risk can be minimized.

Implement Needed DNN Interfaces for Development

One of the final pieces of being a good DNN citizen is to be sure to implement common interfaces that allow your module(s) to take advantage of the most common pieces of functionality within the DNN core. Although not all modules will need to implement each interface, it is important to remember that these interfaces are available and depending on the functionality desired for your module, it might make sense to implement one or more of these interfaces.

The following sections discuss the importance of three of the most common interfaces that are implemented by custom modules. These are considered especially important as they provide critical support features to the framework.

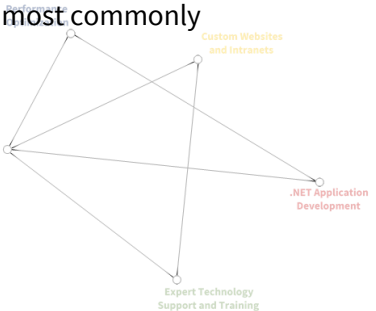
IPortable

The IPortable interface is implemented at the Business Controller class level and is a conduit to allow modules to import and export content. This functionality can be key for many module types. The important items to remember with this interface are that content is exported based on ModuleId. Any other import/export logic would need to be handled by the module in question.

Supporting this interface is critical in situations where individuals are using Portal Templates to setup additional portal sites.

ISearchable

The ISearchable interface is implemented at the Business Controller class level and is a conduit that allows modules to publish information to the DNN search indexer. This is one of the most commonly



forgotten interfaces as the process to implement is not the straightest forward. However, this is typically one of the most needed interfaces as it is what allows users to search and find content entered into a custom module. For implementation assistance we recommend looking at this article: http://www.adefwebserver.com/DotNetNukeHELP/ISearchable/ or the Interfaces section of the "Professional DotNetNuke Module Programming" book.

IActionable

The IActionable interface is implemented at the .ascx user control level and is a method used to add additional actions to the DNN action menu for the module. Implementation of this interface for module actions is considered a Best Practices to provide a consistent set of functionality to users across modules.

As important as it is to implement these interfaces for integration, it is ALSO essential to NOT implement the interface if you have no desire to implement the functionality.

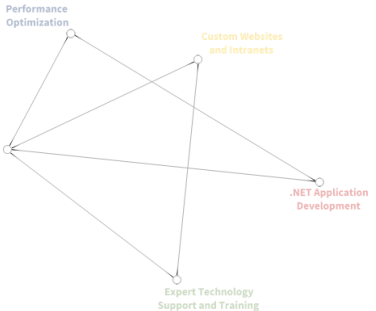
Include License File and Release Notes with Package

When packaging your Extension for installation, it is important to include your software license information as well as release notes so that installing users can see the highlights regarding changes in the most current version. For ease of use these files should be separate files within the solution and referenced as such so that HTML formatting can be used to improve the user experience. The IowaComputerGurus' DNN module templates include this behavior by default for an example.

Cleanup Un-Needed Files

As your Extension grows in maturity, a time will come when you will need to make changes to the file structure and older files will become obsolete. Although not directly harmful to a user's installation, part of being a good citizen is ensuring that you cleanup after yourself.

The DNN manifest supports a "Cleanup" object type which, if used, will remove files from the installation. This will allow you to remove any old DLL's, images, and controls or other files that your application might have created.



Code Specific Requirements

The following section illustrates a few key coding items that should be considered when working with DNN.

Set ValidationGroup Properties on ALL Validators

It is a very common occurrence for a DNN module to be placed on a page with multiple other modules. For this reason, it is important when working with any ASP.NET validator controls that you specify a value for the ValidationGroup. If this property is omitted, you can actually cause entire pages to stop functioning as your validators might be accidentally triggered by an action that occurs outside the scope of your module, specifically, but on the same page in the site.

Testing Scenarios

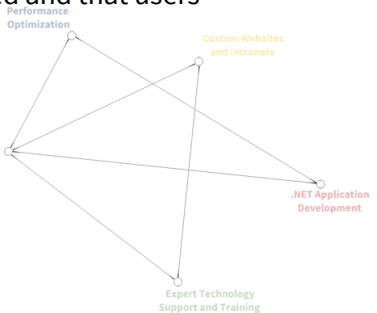
In addition to all of the above scenarios, it is important to note that following a consistent testing process will ensure your Extension has the best ability to play well with the others on an individual DNN installation. At a minimum basis, we have a few recommendations for testing scenarios for any module that will be distributed broadly.

Installation Types

The first item to think about when testing is to ensure that testing is completed both on a Parent Portal (http://www.mysite.com) as well as a Child Portal, (http://www.mysite.com/child) as both of these environments can exhibit different behaviors when creating links and other setup processes.

Test Scenarios

When looking at the two different types of installations, it is important for each release to not only test a 100% clean installation, but also to test an upgrade to ensure that the module properly upgrades and that the instances of the module on the site still function as they should. Un-installation should also be tested with each release to ensure that all data and files are properly removed and that users can re-install the module should they have a need.



Living Document Notice

This Best Practices guide is considered a living document and will be maintained by IowaComputerGurus. It will include the most up-to-date information possible. If you have specific suggestions for content additions, removals, or modifications, please use the contact information found at the end of the document to contact us. We are always looking for validation and additional information that can be added to this document.

About IowaComputerGurus Inc.

IowaComputerGurus Inc, a Microsoft Silver Certified Partner organization, specializes in developing custom solutions using the Microsoft .NET development stack and often utilizing DNN Platform. Based in Ankeny, Iowa, they provide services to customers all over the world and base their business on providing quality, affordable technology solutions with the best customer service. The company is led by Mitchel Sellers, a Microsoft MVP, Microsoft Certified Professional, DNN MVP, and published author.

Contacting IowaComputerGurus

IowaComputerGurus, Inc.

PO Box 737

Ankeny, Iowa 50021

Phone: (515) 270-7063

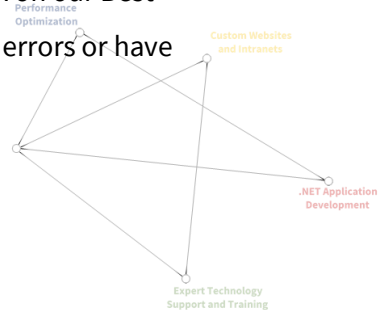
Fax: (515) 866-591-3679

Email: webmaster@iowacomputergurus.com

Website: http://www.iowacomputergurus.com

Feedback

IowaComputerGurus is committed to quality and appreciates constructive feedback on our Best Practices documents, white papers, and other technical guides. If you discover any errors or have



suggestions of additional items for inclusion or any modifications to existing content, please use one of the above listed contact methods to let us know.

Disclaimer

This document reflects the opinions and experience of the authors as a non-compensated service to the community. It is provided "as-is," and no warrantee is expressed or implied as to the serviceability or applicability of any information or recommendation for any particular purpose, application, or environment. It is the reader's responsibility to conduct their own research, consult experts, and determine whether this information is right for their website, application, and / or environment. Additionally, the reader understands use of this documentation constitutes agreement to the current terms of use as published on the IowaComputerGurus.com website, which are subject to change.

Copyright and Trademark Notices