



# **DotNetNuke® Extension Development Best Practices Guide/Recommendations**

**Document Version: 1.00**

**Prepared By: Mitchel Sellers  
Reviewed By: Mitchel Sellers  
Date Created: 08/05/2009  
Date Last Modified: 08/05/2009**

# Table of Contents

<b>Revision History</b>	<b>4</b>
<b>Disclaimer</b>	<b>4</b>
<b>Copyright Notices</b>	<b>4</b>
<b>1. Scope/Purpose of this Document</b>	<b>5</b>
<b>2. Getting Started with Module/Extension Development</b>	<b>5</b>
2.1 Project Templates	5
2.1.1 DotNetNuke Starter Kits	5
2.1.2 BiteTheBullet.co.uk	5
2.1.3 IowaComputerGurus.com	5
2.2 Online Resources	5
2.2.1 Michael Washington's Site ( <a href="http://dotnetnuke.adefwebserver.com">http://dotnetnuke.adefwebserver.com</a> )	6
2.2.2 DnnCreative.com	6
2.2.3 DotNetNuke.com Blogs	6
2.2.4 MitchelSellers.com	6
2.3 Books	6
<b>3. Project Setup/Configuration Recommendations</b>	<b>6</b>
3.1 WAP Vs. WSP Project Structures	6
3.2 Module Folder Naming and Code Namespaces	6
3.2.1 Best Scenario	7
3.2.2 Best Compromise Scenario	7
3.2.3 Namespaces and Assembly Names	7
3.3 Packaging and DNN Version Support	7
3.3.1 Always Set the Minimum Required Version	7
<b>4. Being a Good DotNetNuke Citizen</b>	<b>8</b>
4.1 Support DatabaseOwner and ObjectQualifier	8
4.2 Avoid External Configuration and Web.Config Changes	8
4.3 Re-Use Common DotNetNuke Controls When Possible	8
4.4 Avoid Manipulating DotNetNuke Database Tables Directly	9
4.4.1 Determining Proper APIs	9
4.4.2 Risks of Direct Database Manipulation	10
4.5 Properly Tie Data for Multi-Portal and Multi-Module Installations	10
4.5.1 Foreign Keys and DotNetNuke Tables	10
4.6 Use Consistent Flow for Module Configuration	10
4.7 Carefully Consider any Third-Party Inclusions	11
4.8 Implement Needed DNN Interfaces for Development	11
4.8.1 IPortable	11
4.8.2 ISearchable	11
4.8.3 IActionable	11
<b>5. Code Specific Requirements</b>	<b>12</b>
5.1 Set ValidationGroup Properties on ALL Validators	12

5.2	Use Standard DNN CSS Classes When Possible	12
<b>6.</b>	<b>Living Document Notice</b>	<b>12</b>
<b>7.</b>	<b>About IowaComputerGurus Inc.</b>	<b>12</b>
7.1	Contacting IowaComputerGurus	12
7.2	Feedback	12

## Revision History

Name	Date	Change and Reason For Changes	Version
Mitchel Sellers	08/05/2009	First document created	1.00

## Disclaimer

Information contained in this document has been compiled and is general recommendation information provided by IowaComputerGurus Inc. This information is provided “as-is” and we do not offer any guarantee or warranty on the information provided within. The reader is responsible for performing due-diligence verification that the recommendations in this document fit their needs as well as the specific DotNetNuke environment.

In addition, IowaComputerGurus is a third-party software development firm and is in no way affiliated with DotNetNuke corporation

## Copyright Notices

The information contained within this document is protected under international copyright laws with a content owner of IowaComputerGurus Inc. This document may be re-distributed to anyone, however, it must remain intact and with this disclaimer visible.

The terms “DotNetNuke” and “DNN” are both registered trademarks of DotNetNuke Corporation, <http://www.dotnetnukecorp.com>

## **1. Scope/Purpose of this Document**

This document has been created to provide a baseline set of information relating to DotNetNuke Extension development. 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 obtained over the 150+ custom development projects that have been completed by IowaComputerGurus in the past 3 years.

Not all aspects of each section will apply to every project. In addition there are specific solutions that even with the best of intentions must deviate from the practices listed in this guide. Therefore we want to stress the importance that this is simply a set of basic best practices that should apply in most situations. Additionally this document is NOT intended to be a “how to develop DotNetNuke modules” document. Section two provides information to community based and other resources that might be of benefit for those just starting out with DotNetNuke development.

## **2. Getting Started with Module/Extension Development**

As previously explained in section one this guide was not created to provide an introduction to DotNetNuke 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 sections provide links and information to helpful sites for obtaining project templates, tutorials and books relating to the “getting started” activities with DotNetNuke development. These are just a small sampling of the resources that are available.

### **2.1 Project Templates**

Depending on your language of choice you are presented a few options when it comes to obtaining project templates for creating DotNetNuke modules.

#### **2.1.1 DotNetNuke Starter Kits**

If you are a VB.NET developer or looking for dynamic module templates the DotNetNuke Starter Kit packages are the best place to get your templates. You can obtain these downloads from the [www.dotnetnuke.com](http://www.dotnetnuke.com) website in the form of a simple .vsi installation package.

The starter kit comes with dynamic module (WSP) templates for both C# and VB as well as a compiled VB module template. Please see section 3.1 for recommendations on the two types of projects.

#### **2.1.2 BiteTheBullet.co.uk**

This website contains Visual Studio 2005 and 2008 compiled module templates for those wishing to use C# as their programming language. These templates are single project templates and are the primary foundation that IowaComputerGurus uses for each of our projects.

#### **2.1.3 IowaComputerGurus.com**

After developing multiple projects we identified a small gap in the way that existing templates worked and that is once experienced with module development the “default” code that was included in the templates was not needed and we were spending 15 minutes removing code that should never be in the template. As such, we created C# templates for Visual Studio 2005 that only have project structure and no default code, these templates help to reduce time necessary to go from project creation to development.

### **2.2 Online Resources**

There is a plethora of information available online regarding DotNetNuke 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.

### 2.2.1 Michael Washington's Site (<http://dotnetnuke.adevwebserver.com>)

This tutorial site hits the top of our list as Michaels information is what was used when IowaComputerGurus first started in with DotNetNuke development and the information and tutorials available here provide information that can really help to get someone up and going very quickly. One added benefit of this site is that it does also provide many examples of using new technology with DNN such as Silverlight and LINQ.

### 2.2.2 DnnCreative.com

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

### 2.2.3 DotNetNuke.com Blogs

The blogs on DotNetNuke.com are one of the most commonly overlooked information sources available. Often you will find core team members blogging about new functionality that has/will be included in the core as well as many basic development tutorials are listed.

### 2.2.4 MitchelSellers.com

In addition to the above resources Mitchel will often blog about various DotNetNuke and .NET development topics on his blog.

## 2.3 Books

There are many DotNetNuke books out on the market. There is only primary book relating to DotNetNuke module development, "Professional DotNetNuke Module Development", written by Mitchel Sellers. This book covers the foundation of DotNetNuke module development and is a great starting point. Some of the other DotNetNuke books cover development as well with varying levels of detail.

## 3. Project Setup/Configuration Recommendations

The first section of "best practices" recommendations support the actual project setup, configuration, and packaging for deployment. These recommendations are very general in nature and apply to all types of extension development.

### 3.1 WAP Vs. WSP Project Structures

It has been the experience of IowaComputerGurus that using the Web Application Project (WAP) development model is a much better approach when working with custom or commercial modules when compared to Web Site Projects (WSP). We have come to this conclusion for a number of compelling reasons:

1. WAP allows for easy compilation to a single dll for all code
2. WAP allows for easy source control integration with a solution for each module
3. WAP encourages working with just the DotNetNuke.dll, with a separate solution any attempts at non-standard integration is discouraged simply due to the project structure
4. WAP allows for post-build events that can help automate module packaging and speed up development efforts.
5. WAP allows for creation of multiple modules that have dependencies without installation order risks and other quirks associated with inter-module dependencies with WSP

It is for these reasons that we utilize and strongly suggest the regular usage of the WAP model. Although something that can be a bit tricky at first when it comes to debugging we have found that in the end a more stable development environment is created and developers can focus more easily on the task at hand.

### 3.2 Module Folder Naming and Code Namespaces

One of the most commonly overlooked items by developers is in relation to the name of the folders for their modules and/or the namespaces associated with their .NET code. Developing for DotNetNuke does introduce a few challenges that might not be immediately realized and that is that a DNN user can install a module developed by any company. This 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 minimized greatly. The final point in this section discusses the importance and recommendations around Namespace and Assembly definitions.

### **3.2.1 Best Scenario**

From a best practices standpoint we have found that placing all custom modules developed by a single company into a common folder is one way to easily 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.

This naming convention helps ensure that the only file conflicts that would be introduced are related specifically to code that you deploy. The process of configuring this is slightly involved, requiring the changing of base path and other path values in the respective .dnn manifest files.

### **3.2.2 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 a good place to start. For example with the guestbook placing the module in a ICG\_Guestbook folder would reduce the risk of conflict.

We don’t like this scenario as much simply because it doesn’t group modules together and the use of the underscore character just doesn’t seem to feel as good of a solution.

### **3.2.3 Namespaces and Assembly Names**

Aside from physical file naming within the module folder it is important that namespaces and assemblies be created in a manner that will not conflict with others. Using a namespace structure that starts with Company.Modules.Module name or ICG.Modules.Guestbook for an 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, for this we recommend naming this file the exact same as the root namespace for the project. In our example we would generate a dll with a 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.

## **3.3 Packaging and DNN Version Support**

One of the most common struggles that we see with module developers is how/when to package their module for distribution. With the release of DNN 5.x earlier in 2009 this has become an even harder situation for individuals to handle. On one hand if you package for DNN 5.x exclusively you can take advantage of all the new features that come with the new manifest format, however, you limit yourself to only uses on the 5.x platform with a large number of individuals on the old platform.

You have an additional option of maintaining two packages, but that becomes very troublesome and hard to manage even after a single release. Therefore it is our recommendation to build the module against, and package the module against the lowest supported version. For example all of our modules are built against 4.7.0 to support 4.7.0 and later. Yes, this limits us a bit, but it ensures the widest audience possible for our modules.

### **3.3.1 Always Set the Minimum Required Version**

Associated with the package decision is ensuring that if a user tries to install your module on a non-supported version that it does not cause them problems. If you build your module against DNN 4.7.0 and a user tries to install it to a DNN 4.6.0 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.

To ensure that this does not happen all DotNetNuke manifests support the specification of a version limiter, for example adding the following to a 4.x manifest will ensure that it only installs on 4.6.0 and later.

```
<compatibleversions>^[0-9]{1}[4]{1}.[0-9]{1}[6-9]{1}.[0-9]{1}[0-9]{1}||[0-9]{1}[5-9]{1}.[0-9]{1}[0-9]{1}.[0-9]{1}[0-9]{1}$</compatibleversions>
```

By doing this, any attempted installation to a prior version will fail, and the users site will still function properly. This can help reduce support calls substantially.

## 4. Being a Good DotNetNuke Citizen

Section three of this document talked about the project structure and the beginnings of what it takes to build an extension that will play well inside the DotNetNuke environment. This section of the document takes the foundation of this a bit further and talks about practices that should be followed to ensure that developed solutions play well across DotNetNuke sites with many different configurations.

### 4.1 Support DatabaseOwner and ObjectQualifier

One key configuration element of DotNetNuke that is often overlooked by module developers is the ability to specify custom DatabaseOwner and ObjectQualifier values. Although in our experience not heavily used items, omitting them 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 that it does not make sense to not support them. We have found that if 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.

### 4.2 Avoid External Configuration and Web.Config Changes

Although with the new manifest file format provided in DotNetNuke 5.x it is easier for developers to make web.config changes without requiring that site administrators manually make changes, it is our recommendation that solutions avoid any web.config changes. By avoiding these configurations you have a module that will install in a more streamlined approach and there are less moving parts that can go wrong.

Even with the XmlMerge functionality that comes with 5.x 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.

### 4.3 Re-Use Common DotNetNuke Controls When Possible

One of the biggest advantages of leveraging the DotNetNuke platform is the plethora of built in controls, tools, utilities and functions that can be used. Using these built in defaults users will be presented with interfaces that are familiar to them as well as can reduce your overall time to development. A listing of a few of the most common controls and their respective locations are included for reference.

Control Name	Path	Functionality
Label	~/Controls/LabelControl.ascx	Provides a label and expanding help text icon
TextEditor	~/Controls/TextEditor.ascx	Provides rich text editor, using the provider configured in the web.config. Default provider FCK editor.
UrlControl	~/Controls/UrlControl.ascx	Provides functionality for Url, Page, or file selection as well as the ability to upload files within the existing DotNetNuke file structure. Used across the DNN

		installation by default.
SectionHeader	~/Controls/SectionHead.ascx	This is an expanding/collapsing section header control, the same control that is used for the various sections within the “Admin Settings” and “Host Settings” pages.
Various	DotNetNuke Web Controls Library	Inside this assembly there are a number of additional controls such as the PropertyEditorControl, ProfilePropertyEditor, and DNN Paging Control. Many of which can be highly helpful for development activities.

If you are unfamiliar with the usage of any of these controls, or the others that are not included in the table above please see the various resources listed in section one, as many implementation examples can be found via those resources.

#### 4.4 Avoid Manipulating DotNetNuke Database Tables Directly

One of the biggest mistakes that we see when reviewing custom solutions that have been developed for customers in the past are cases where existing DotNetNuke API’s are not leveraged and developers are either re-creating the wheel or they are actually manipulating DNN database tables and data directly. As part of this, for our best practice recommendation we say that you should AVOID doing anything to DNN tables directly, and to help understand how to compensate for this, the following sub-sections will give some needed insight to where you can find the API’s necessary to leverage the DNN functionality as well as additional information on the reasons why direct database access is risky.

##### 4.4.1 Determining Proper APIs

Sadly, one of the areas that are lacking within the DotNetNuke framework is in 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 into key DotNetNuke data that is commonly needed in custom solutions.

<i>Namespace</i>	<i>Description</i>
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 DotNetNuke 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 DotNetNuke 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 provider information on users within the DotNetNuke portal. The UserController provides most 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 DotNetNuke does expose a very robust API for programming. The only real limiting factor is that sometimes you may have to play with a method just to see

what it does due to the lack of API comments.

#### 4.4.2 Risks of Direct Database Manipulation

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

##### 4.4.2.1 Upgrade Risks

The primary risk with direct database manipulation comes with future upgrades to DotNetNuke. There is no set criterion that prevents DotNetNuke 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 very hard to quantify, however, is something to be concerned about, especially with the massive overhauls and improvements that have been coming with the 5.x series.

##### 4.4.2.2 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 however the respective cache entries are purged and the information is updated

#### 4.5 Properly Tie Data for Multi-Portal and Multi-Module Installations

Another best practice area surrounds the whole concept of data isolation. What DotNetNuke data element should drive the storage of your data? Our best practice recommendation in this area is mostly focused on using common sense, the following table outlines the three most common data elements used:

<i>DotNetNuke Data Element</i>	<i>Use For Storing</i>
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 is 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 is just to be sure to use appropriate options.

##### 4.5.1 Foreign Keys and DotNetNuke Tables

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

It has been our experience that the latter is the true best practice as it helps keep old data from cluttering up a system is a module is deleted and 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 DotNetNuke system.

#### 4.6 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 practice 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, and 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.

#### **4.7 Carefully Consider any Third-Party Inclusions**

One of the most important 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 reason for this is that with DNN 4.x, there is not support in DNN to control if/when the dll should be added or updated from the system. Now, if your custom solutions are the only items installed on the site, you do not have a lot of risk. However, say for example that you utilize something similar to the Telerik controls in a custom module, and the site also has a skin that uses the Telerik Rad Menu. If you include the Telerik dll with your module and the administrator removes the module selecting the "Delete Files" button the Telerik dll will be removed as well, rendering the site unusable.

Now, if you are developing specifically for DotNetNuke 5.x, this risk can be mitigated by using the assembly component type, with this component type DNN will manage references to the assembly and prevent this type of issue.

#### **4.8 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 DotNetNuke 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.

##### **4.8.1 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 is 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.

##### **4.8.2 ISearchable**

The ISearchable interface is implemented at the Business Controller class level and is a conduit that allows modules to publish information to the DotNetNuke search indexer. This is one of the most commonly forgotten interfaces as the process to implement is not necessary the most straight 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.adeftwebserver.com/DotNetNukeHELP/ISearchable/> or the Interfaces section of the "Professional DotNetNuke Module Programming" book.

##### **4.8.3 IActionable**

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

## 5. Code Specific Requirements

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

### 5.1 Set ValidationGroup Properties on ALL Validators

It is a very common occurrence for a DotNetNuke 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.

### 5.2 Use Standard DNN CSS Classes When Possible

Another best practice recommendation when it comes to custom code is to utilize "standard" DotNetNuke CSS Classes for elements whenever possible. By following this practice it is very easy for custom modules to work well with existing skins and to take on the various skin elements of the destination site. Some of the most common classes to observe and follow are listed below. Note, this is simply a small sampling of the most common elements.

CSS Class	Usage
Normal	This is the CSS class for standard text within a site/module
NormalRed	This is the CSS class for error messages, typically the normal font with a red color
Head	This is for major headings within the site
SubHead	This is for input labels and items of similar nature.

## 6. Living Document Notice

This best practices guide is considered a living document and will be maintained by IowaComputerGurus and updated to 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.

## 7. About IowaComputerGurus Inc.

IowaComputerGurus Inc, a Microsoft Certified Partner organization specializes in developing custom solutions using the Microsoft .NET development stack and often using the DotNetNuke application framework for web application development. Based in Des Moines, 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 lead by Mitchel Sellers a Microsoft MVP, Microsoft Certified Professional and published author.

### 7.1 Contacting IowaComputerGurus

IowaComputerGurus Inc.  
217 East Porter Avenue  
Des Moines, Iowa 50315

**Phone:** (515) 270-7063

**Fax:** (866) 591-3679

**Email:** [webmaster@iowacomputergurus.com](mailto:webmaster@iowacomputergurus.com)

**Website:** <http://www.iowacomputergurus.com>

### 7.2 Feedback

IowaComputerGurus is always looking for feedback on our Best Practices guides. If you have suggestions of additional items for inclusion or any modifications to existing content please use one of the above listed contact methods and we will be sure to update the document accordingly.