File System Abstraction and Folder Providers in DotNetNuke

Brandon Haynes

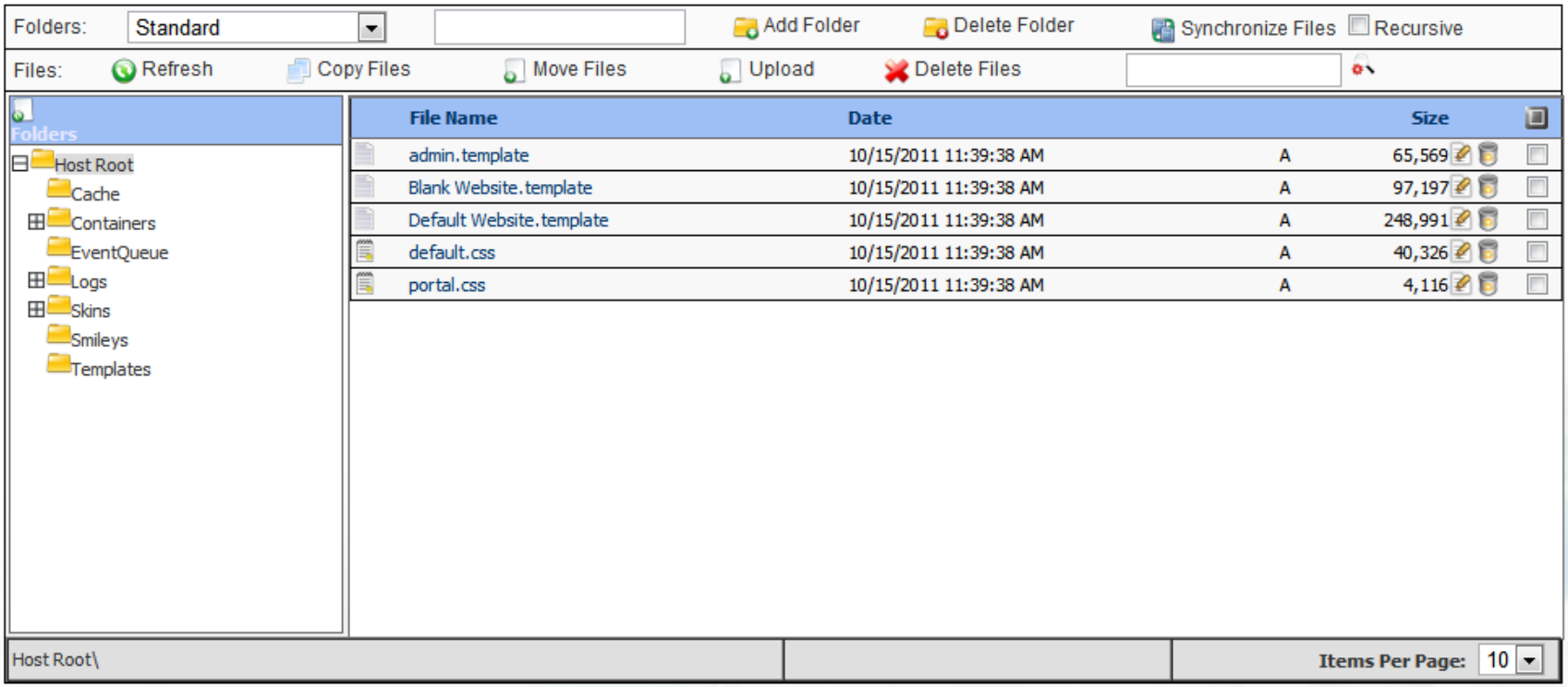
Agenda

- Background
  - File Services in DotNetNuke 5 and 6
- Architecture
  - File and Folder Managers
  - Provider Model and Folder Providers
  - Core Implementations
- Extension
  - Why?
  - Hello World
  - Build something cool

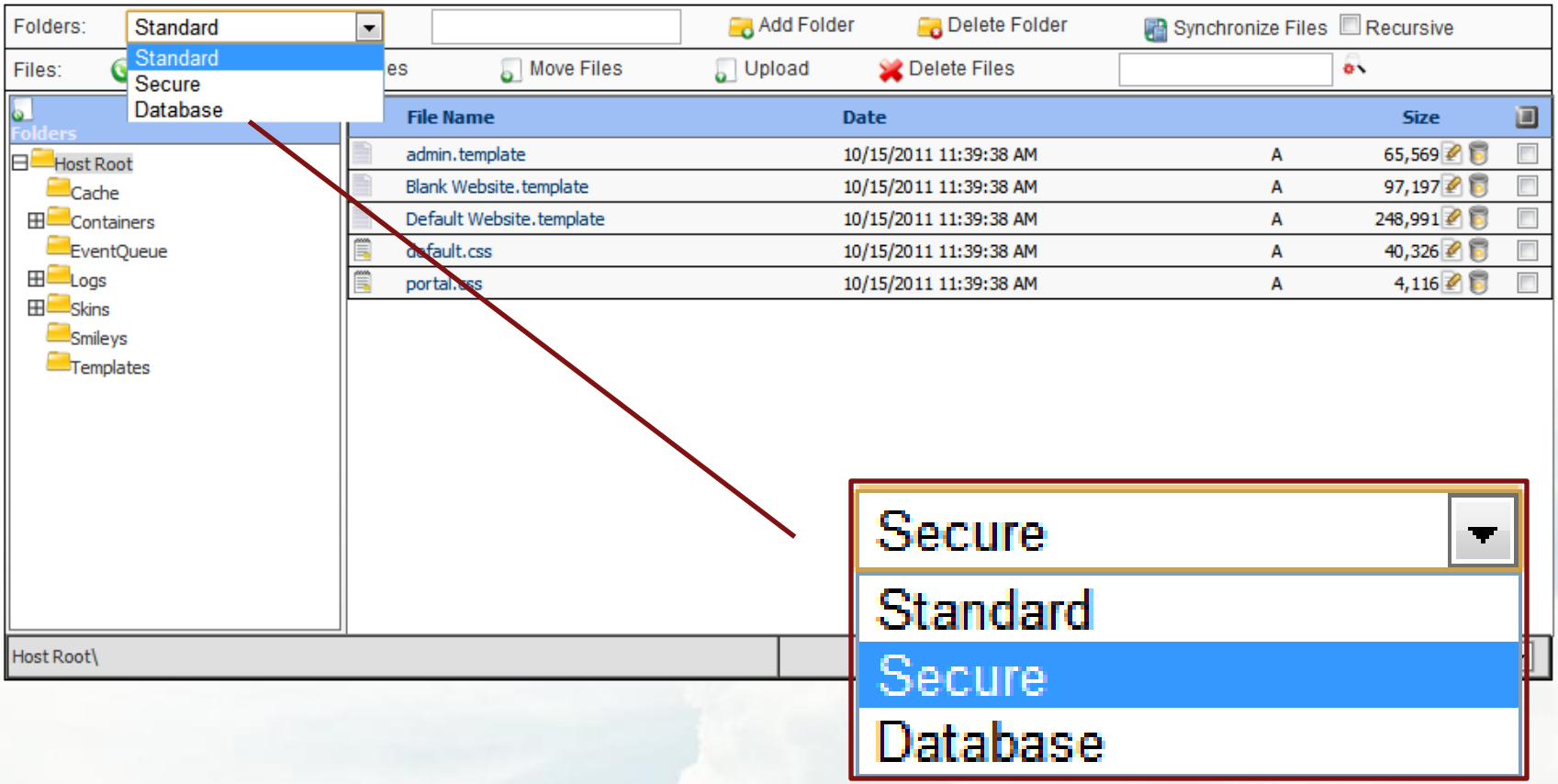
Background

File System Abstraction and Folder Providers in DotNetNuke

The File Manager



The File Manager



File Types in DotNetNuke 5.x

- `FolderController.StorageLocationTypes`
  - `InsecureFileSystem` (Standard)
  - `SecureFileSystem` (Secure)
  - `DatabaseSecure` (Database)

File Types in DotNetNuke 5.x

- `InsecureFileSystem` (Standard)
  - Exists on the local file system
  - May be directly accessed via URI
  - Path obscurity via LinkClick HTTP Handler

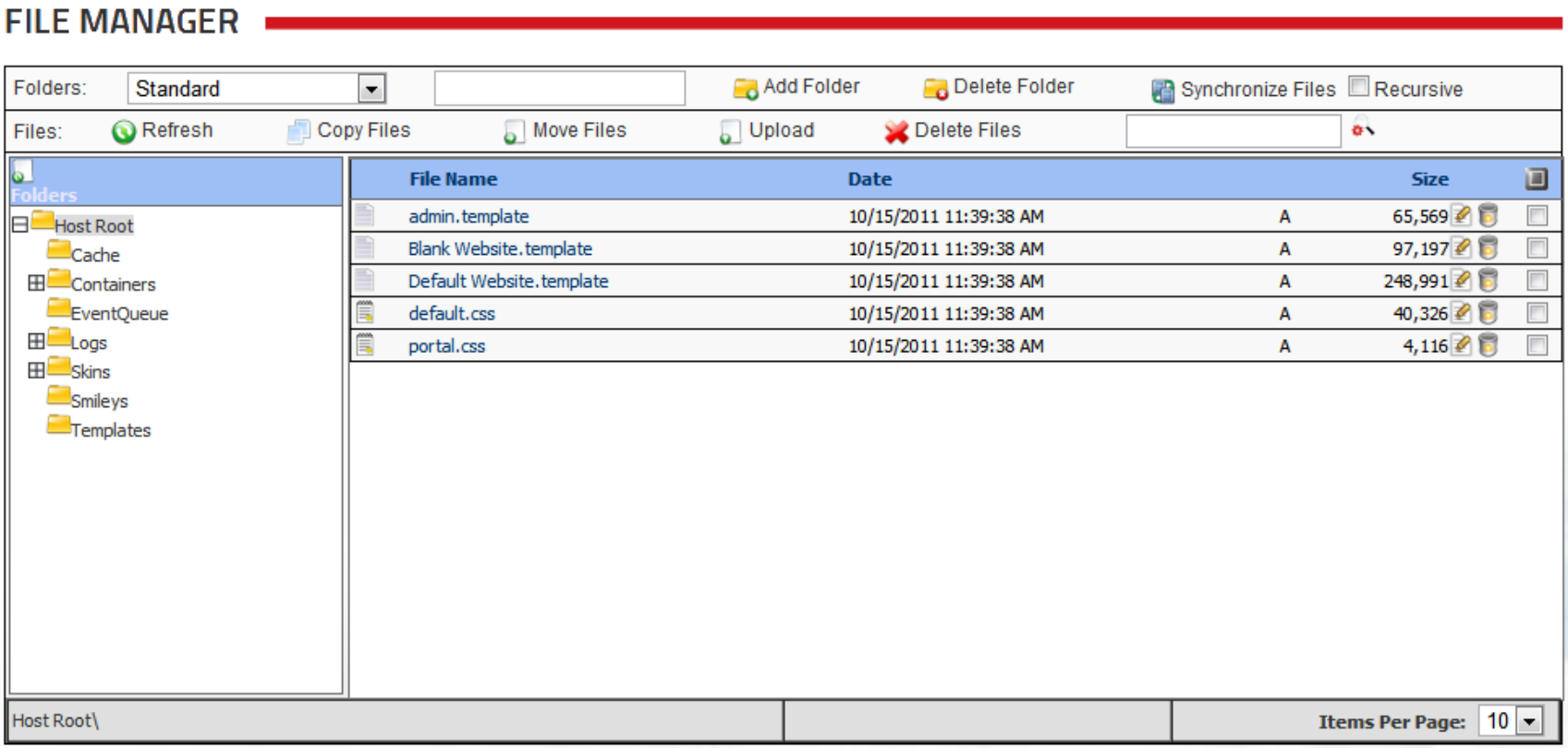
File Types in DotNetNuke 5.x

- `SecureFileSystem` (Secure)
  - Exists on local file system
  - ".resources" extension appended
  - Not served by IIS (by default)
  - Streaming access via LinkClick HTTPHandler

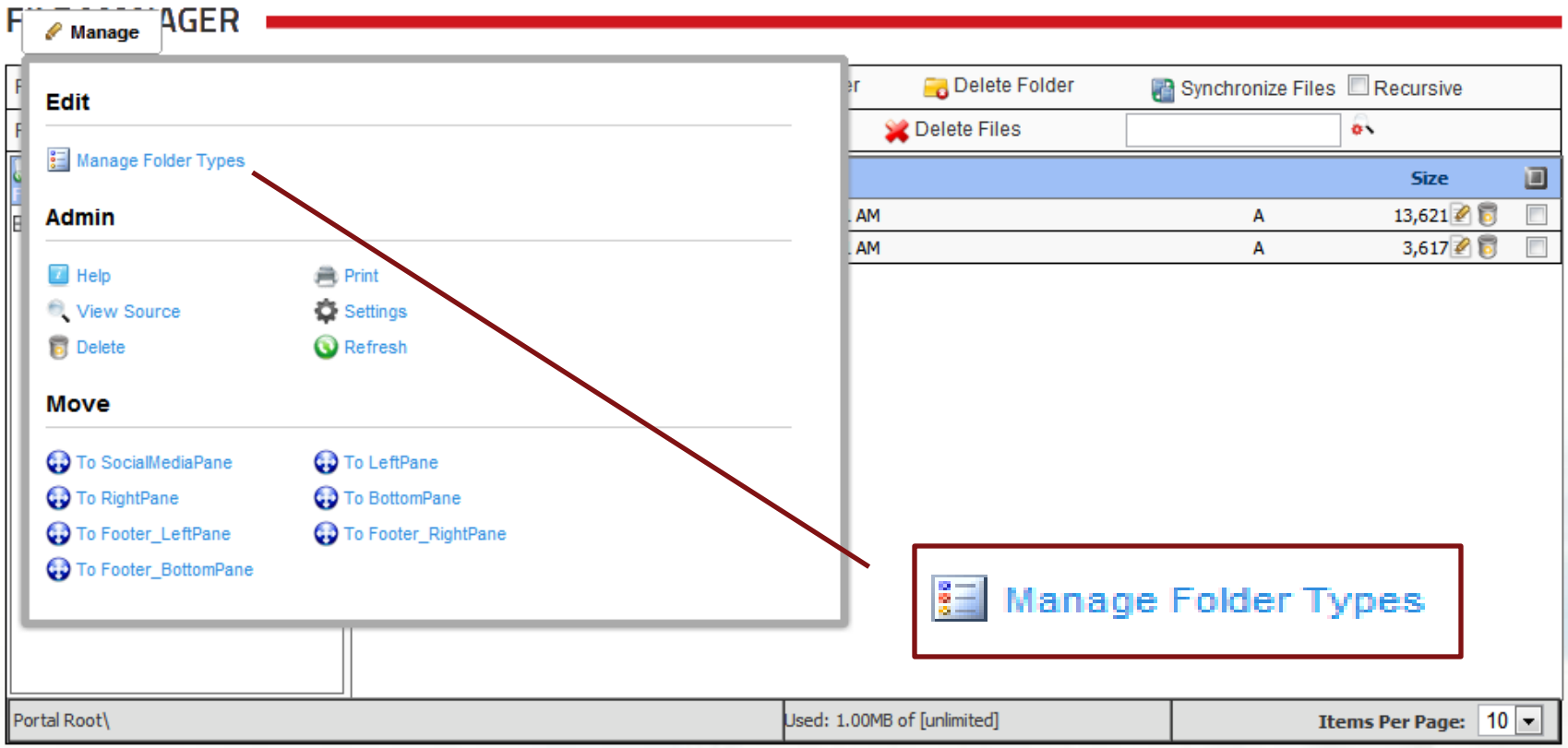
File Types in DotNetNuke 5.x

- `DatabaseSecure` (Secure)
  - Persisted in database
    - In the Files table, Content field
  - Streaming access via LinkClick HTTPHandler
  - Performance considerations

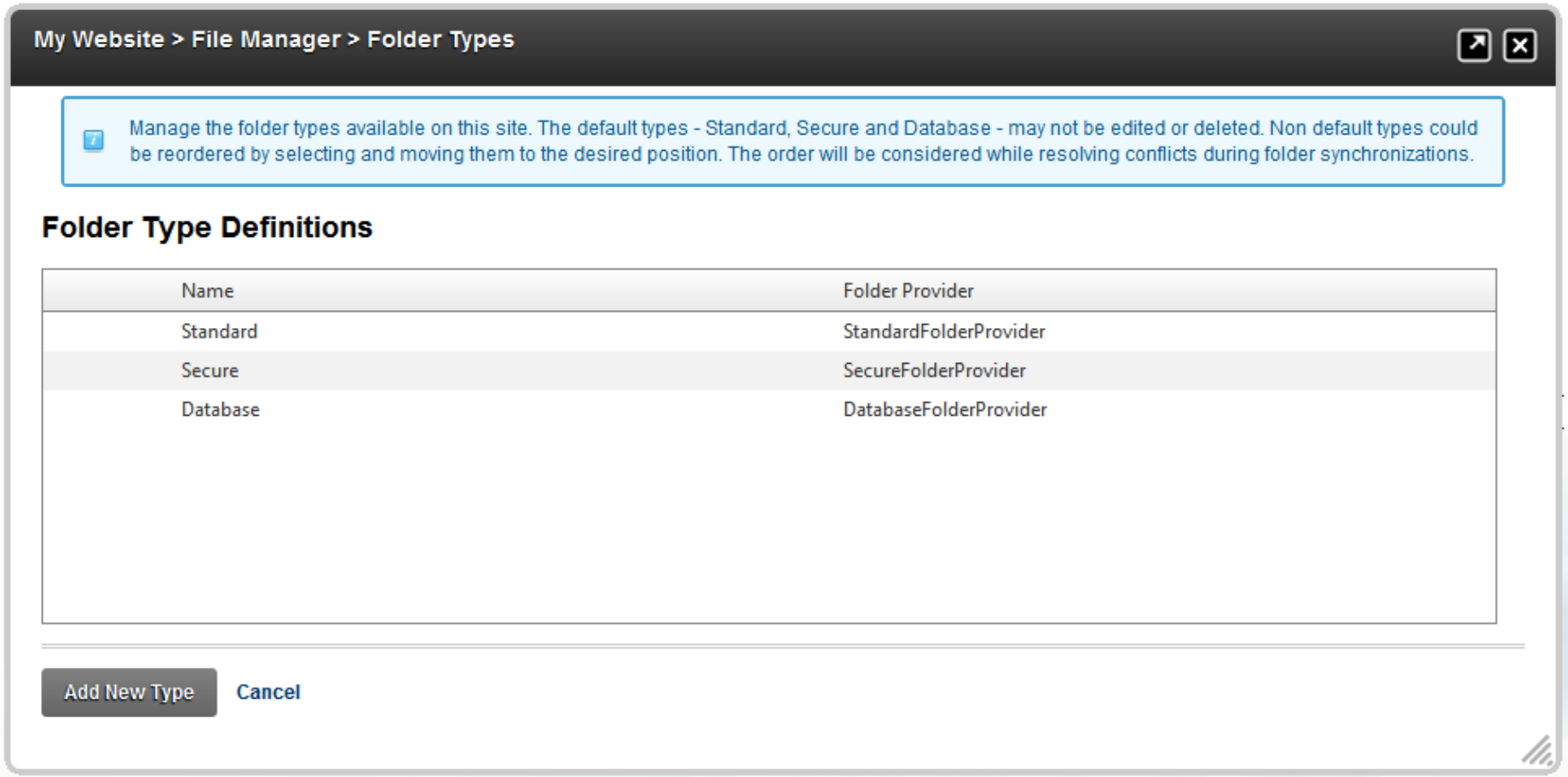
File Manager, DotNetNuke 6.x



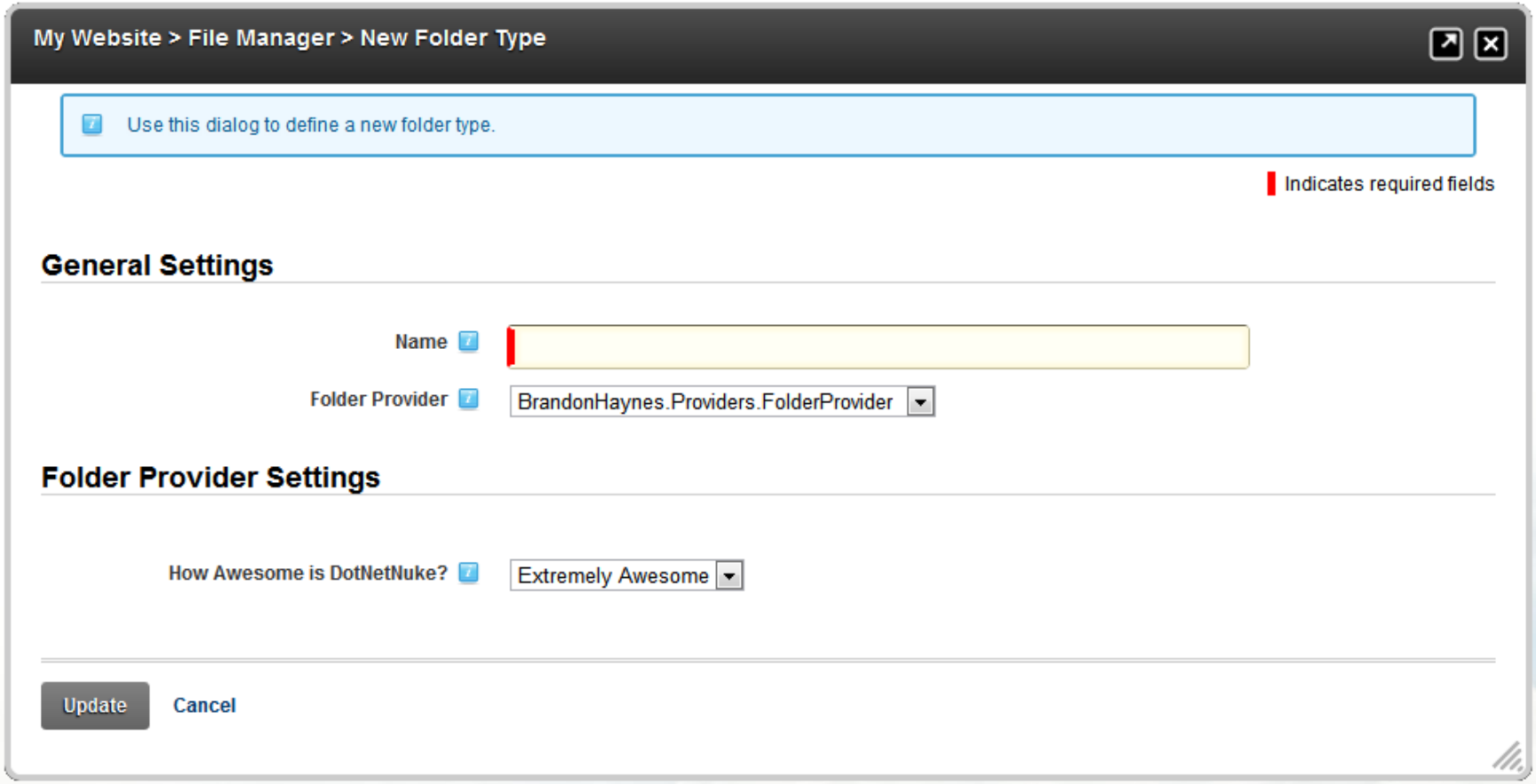
File Manager, DotNetNuke 6.x



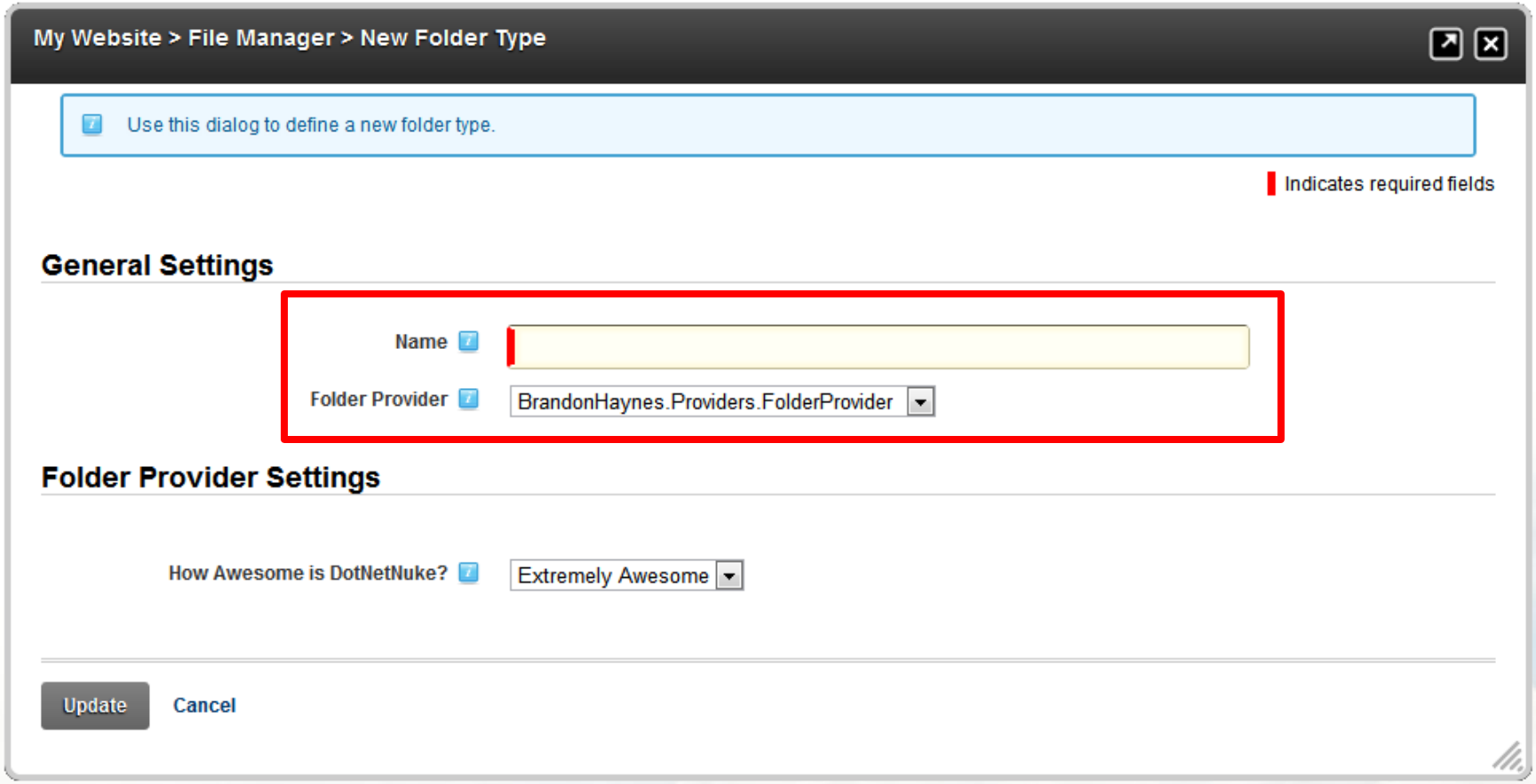
File Manager, DotNetNuke 6.x



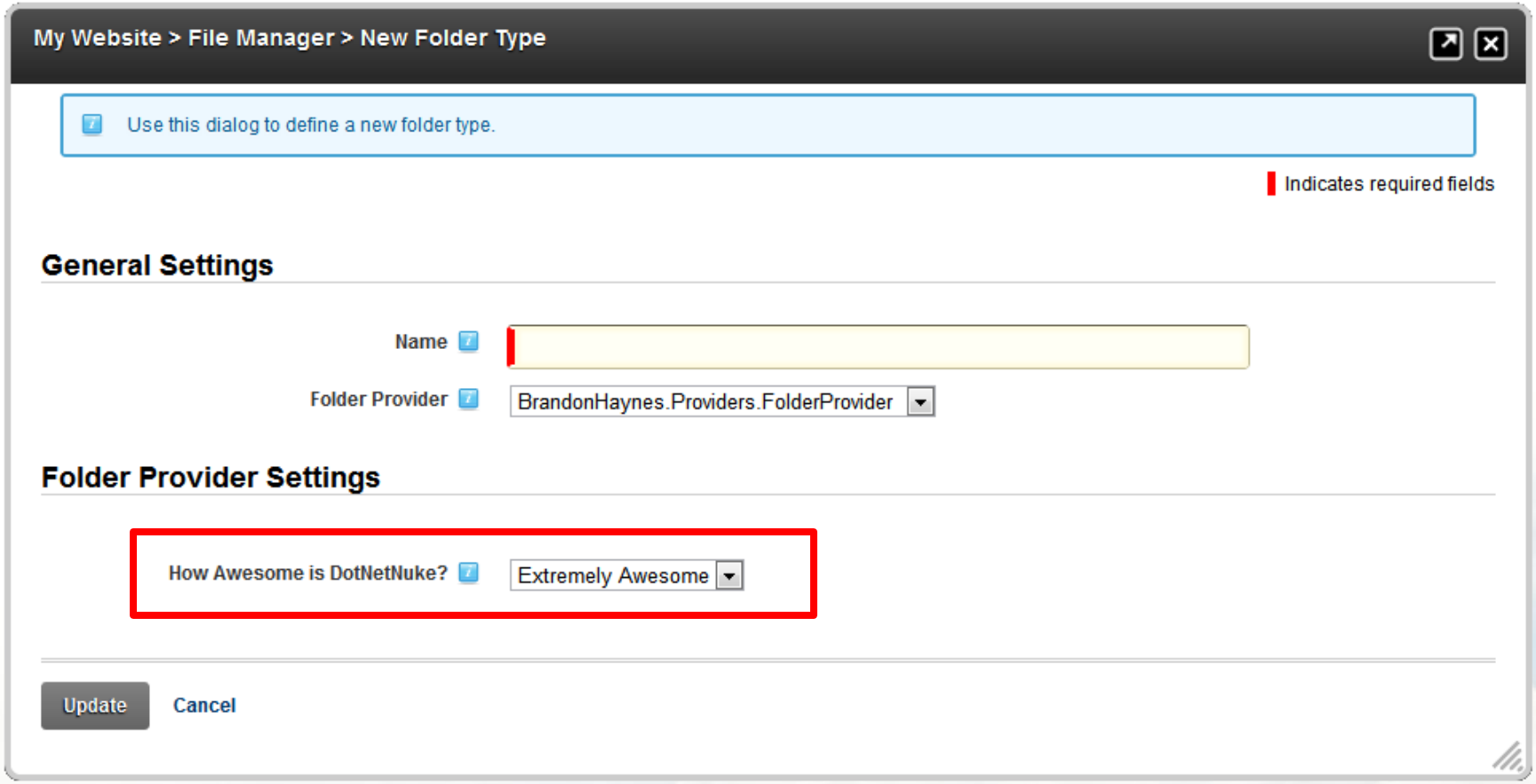
File Manager, DotNetNuke 6.x



File Manager, DotNetNuke 6.x



File Manager, DotNetNuke 6.x



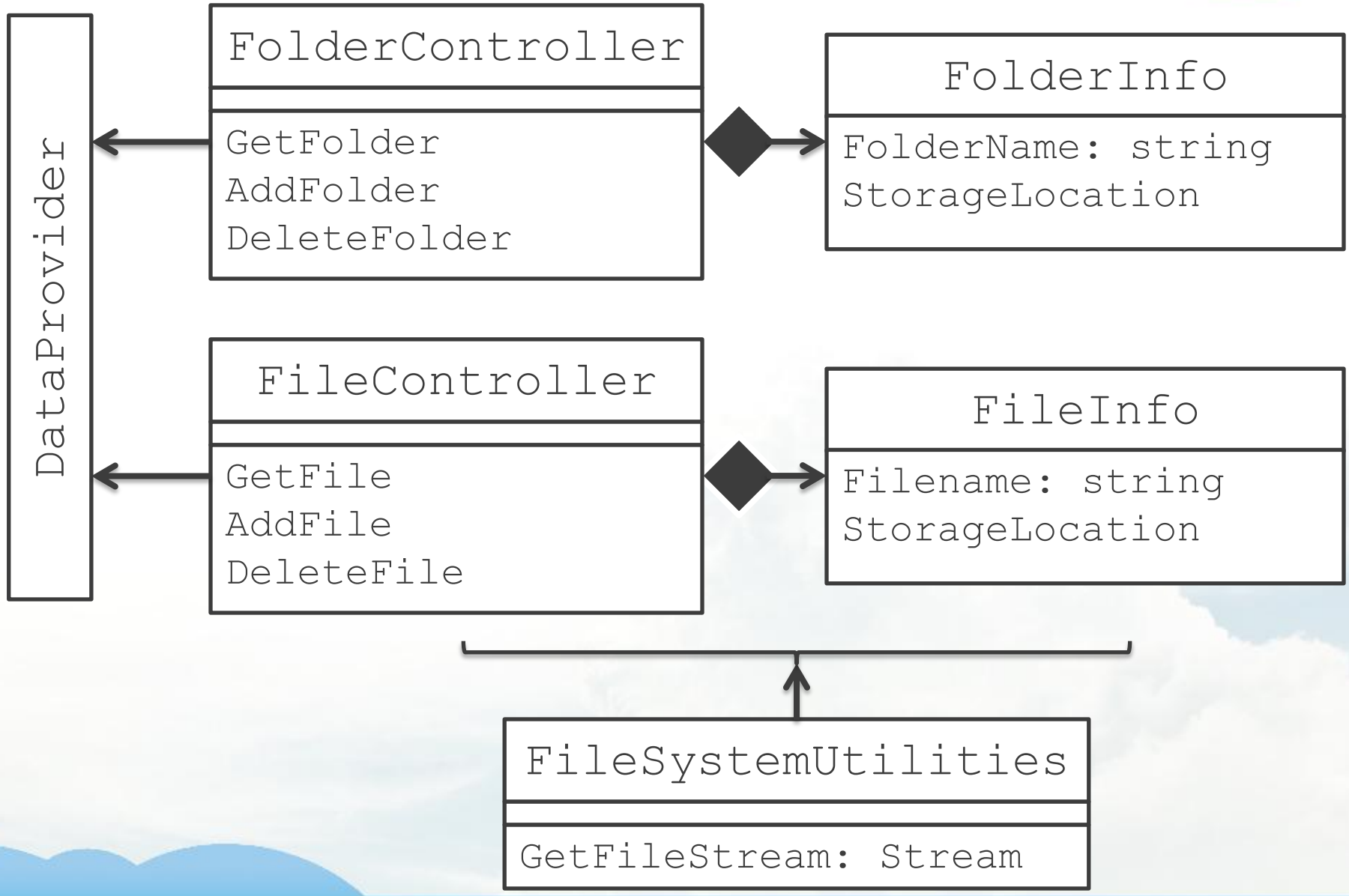
Core Folder Provider Implementations

- Community (DotNetNuke.Services.FileSystem)
  - StandardFolderProvider
  - SecureFolderProvider
  - DatabaseFolderProvider
- Professional
  - Amazon S3
  - Windows Azure
  - SharePoint
- Others
  - Evotiva DNNGlobalStorage (Amazon S3)
  - Intelequia DNNFolderProviders (Windows Azure)
  - Sharepoint Folder Provider (Alpha)

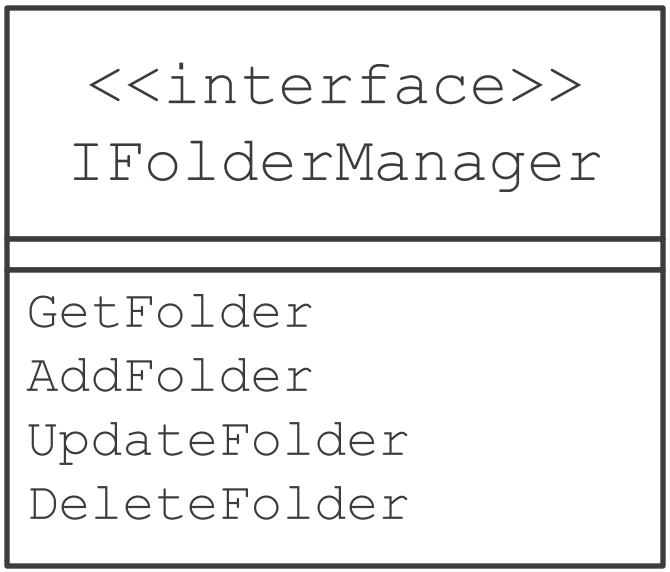
Architecture

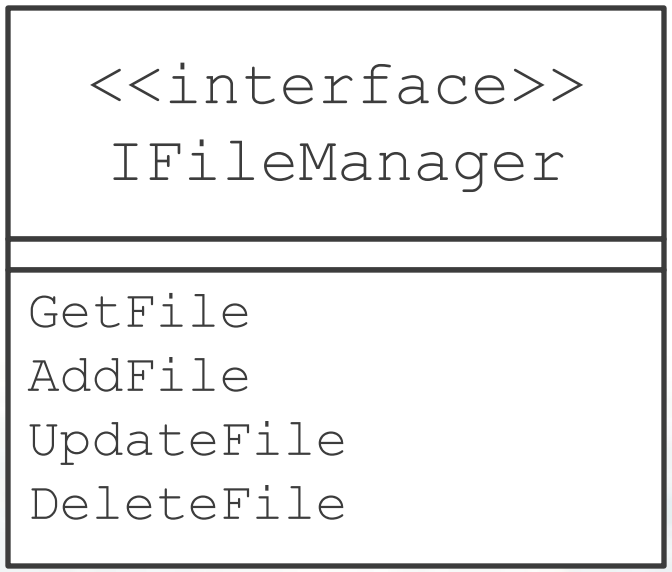
File System Abstraction and Folder Providers in DotNetNuke

DotNetNuke File System (5.x)

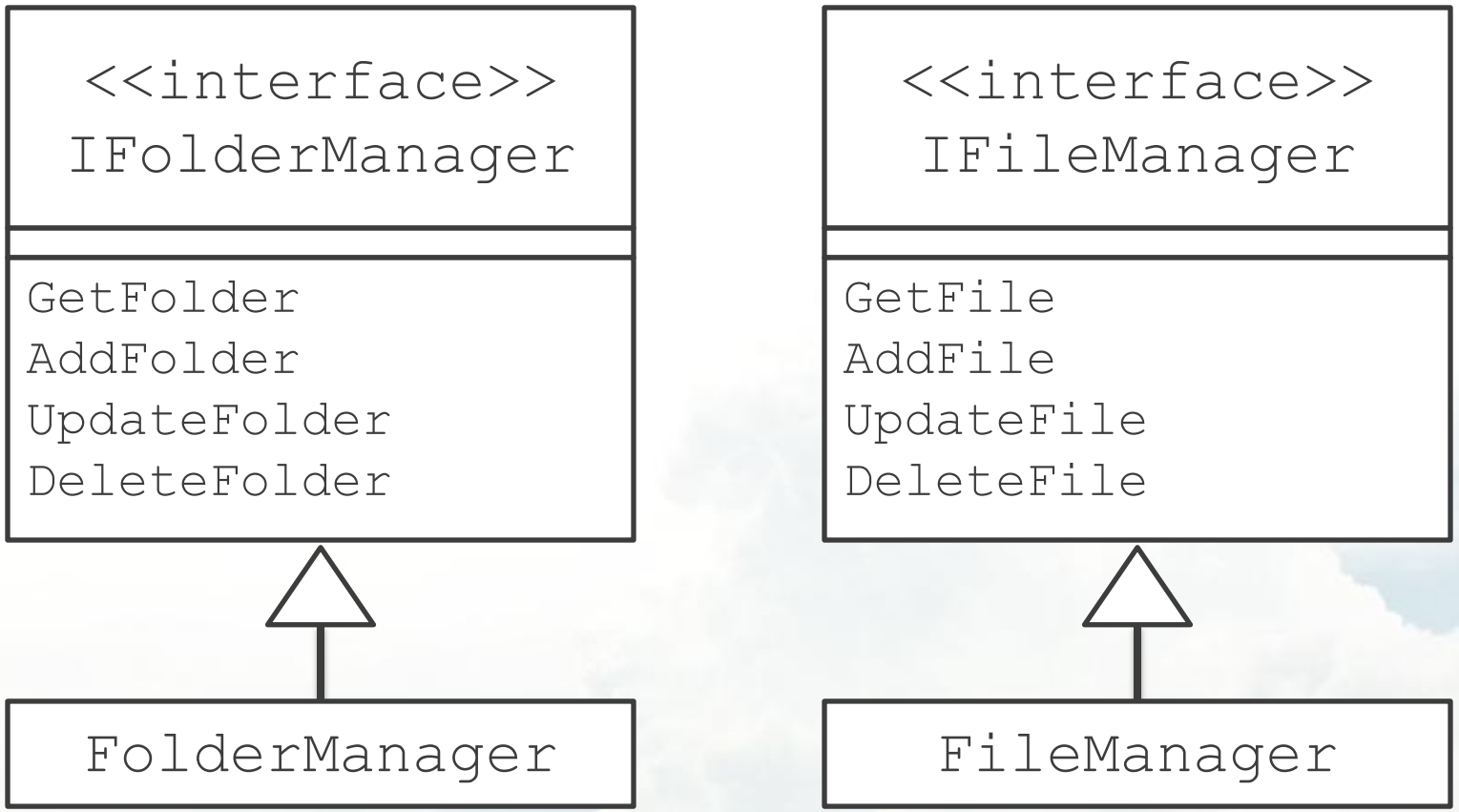


DotNetNuke File System (6.x)

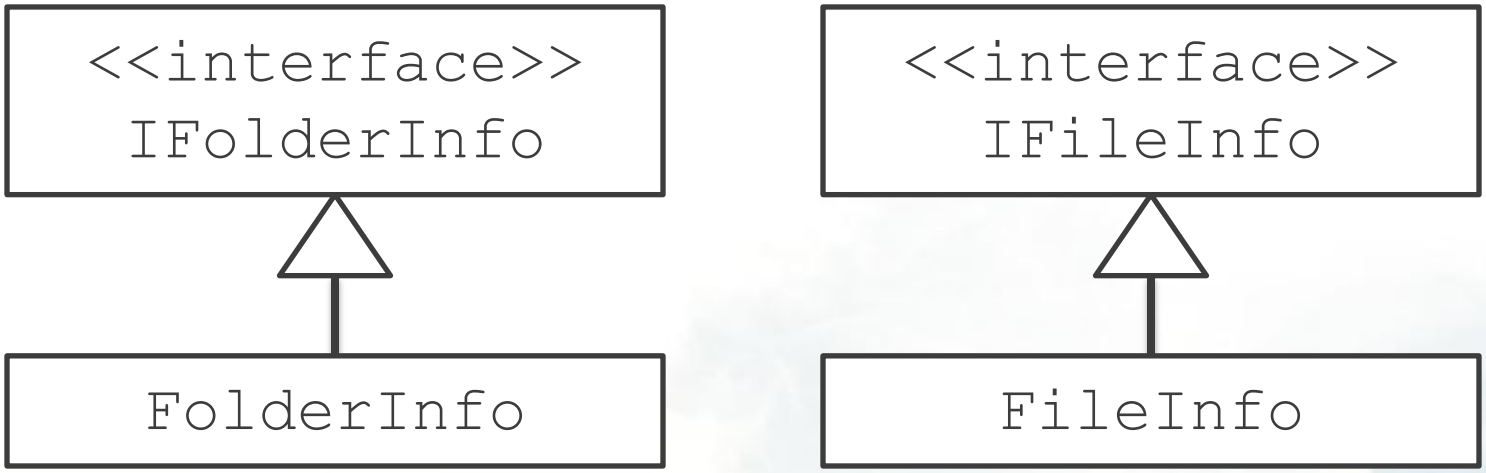




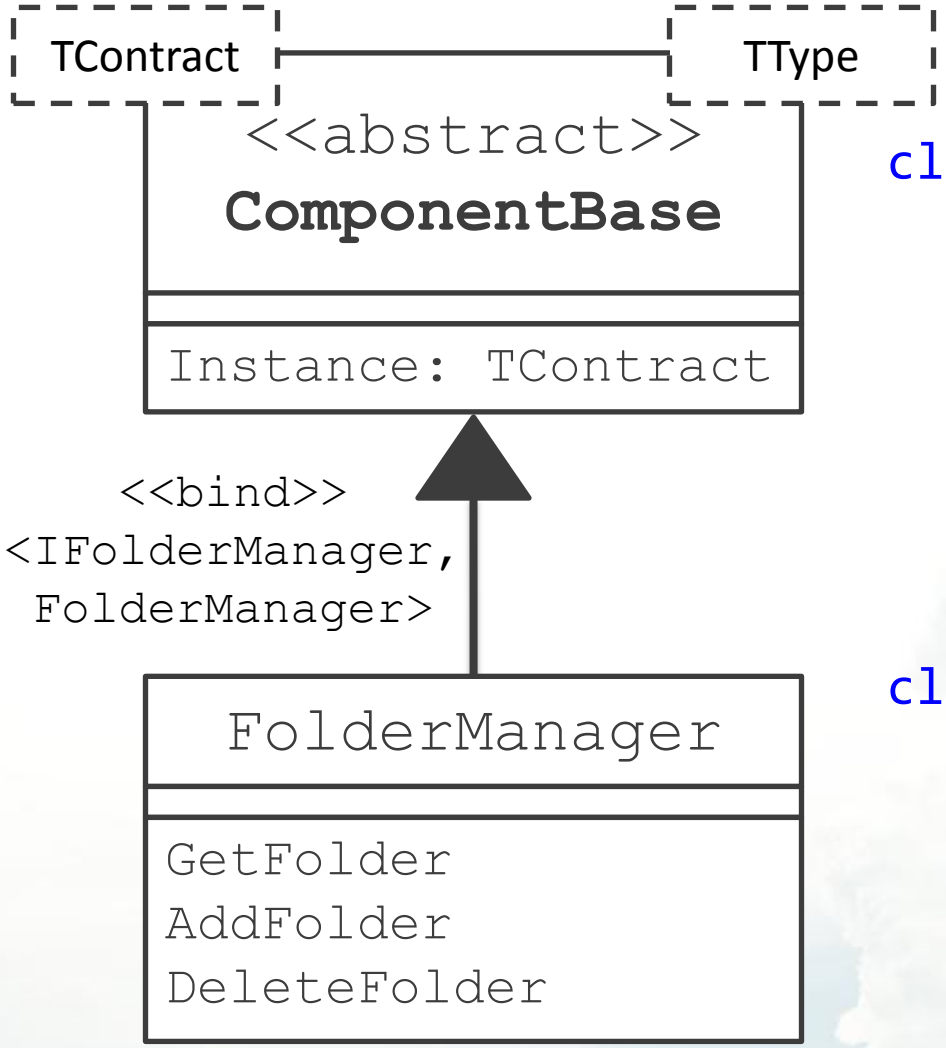
DotNetNuke File System (6.x)



DotNetNuke File System (6.x)



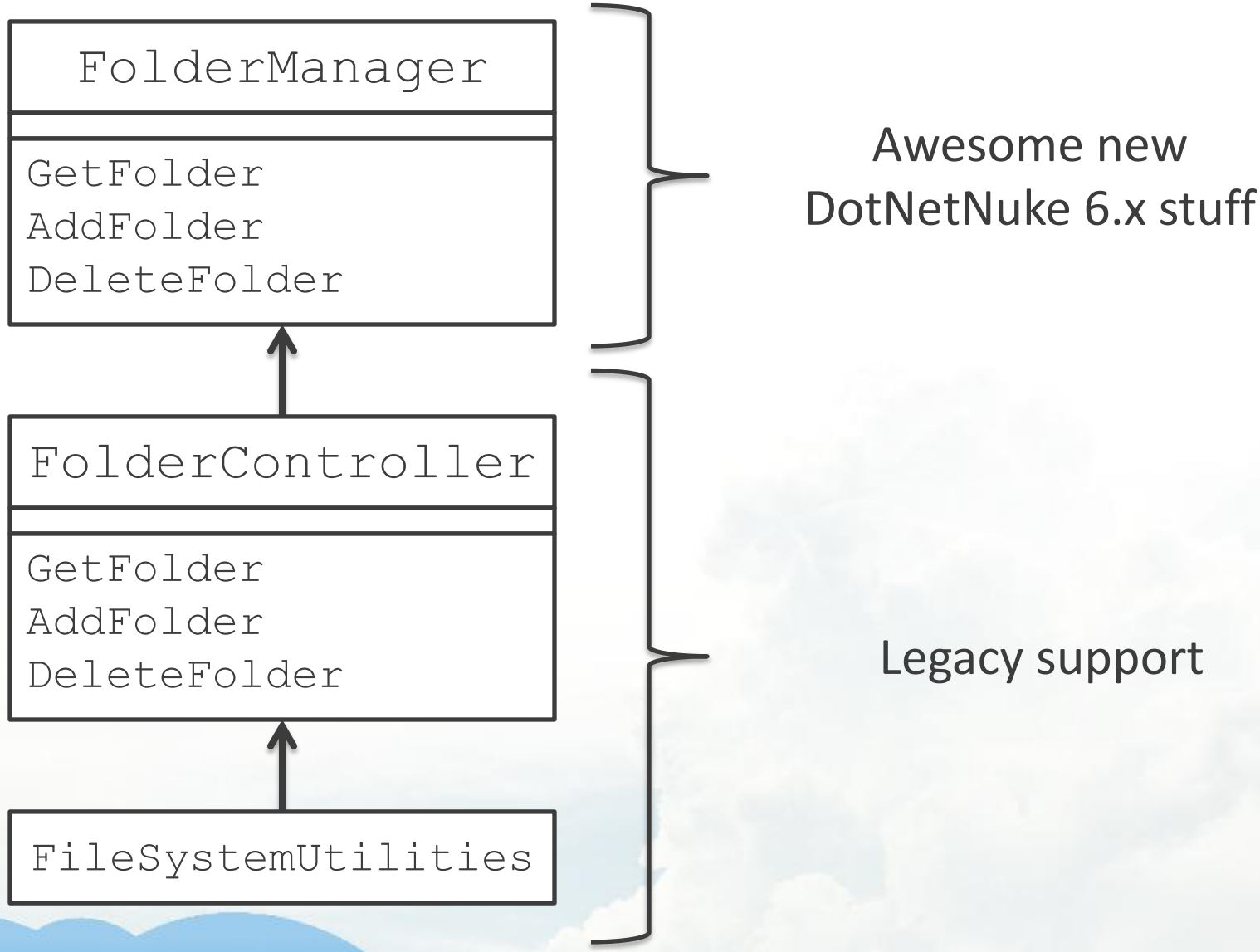
DotNetNuke File System (6.x)



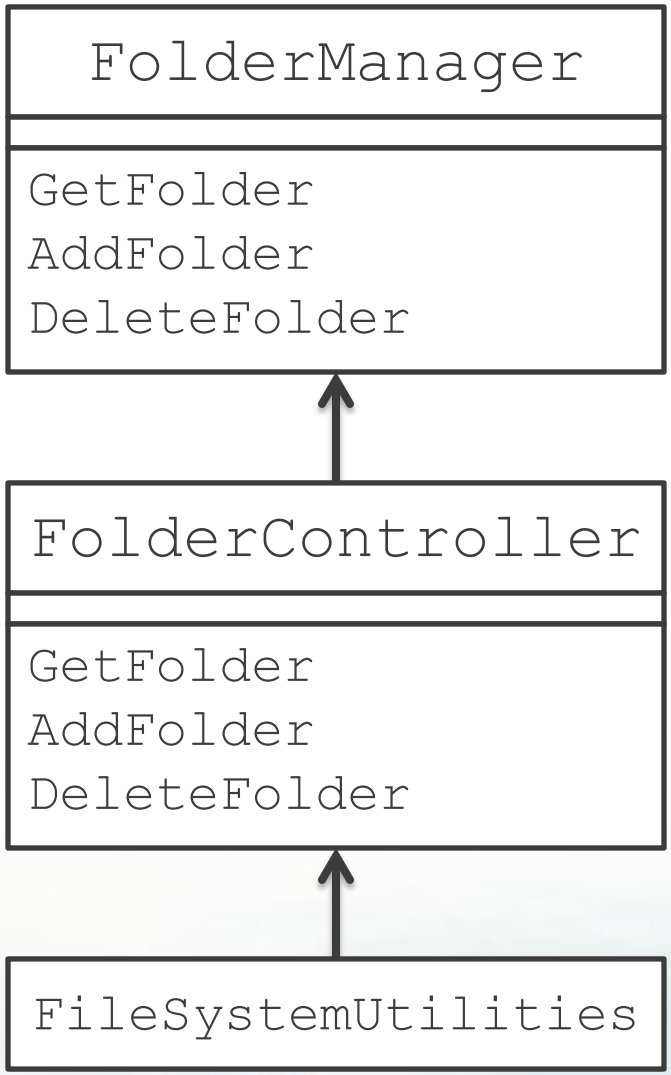

```
class ComponentBase<TContract,
                    TType>
  where TType : TContract
  {
  static TContract Instance
      { get { ... } }
  }

class FolderManager
  : ComponentBase<IFolderManager,
                  FolderManager>,
    IFolderManager
  {
  ...
  }
```

DotNetNuke File System (6.x)

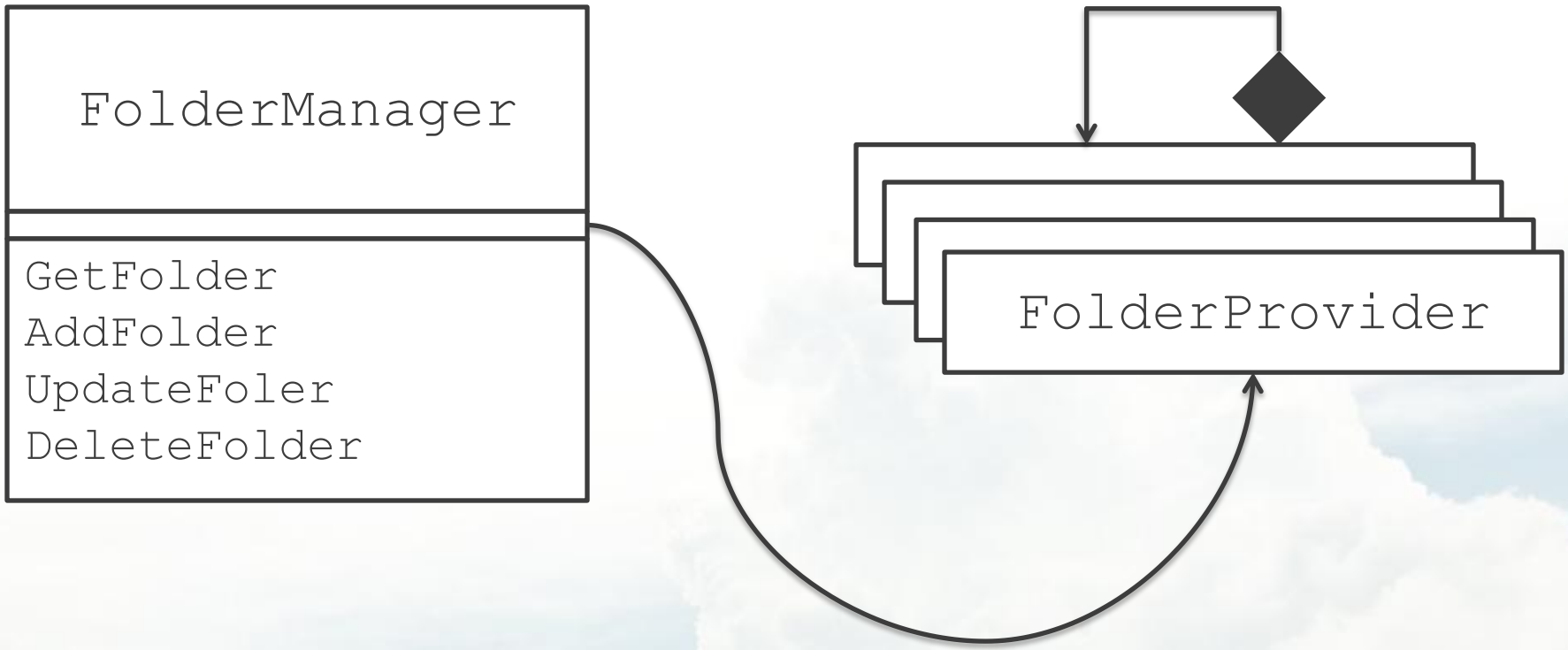


DotNetNuke File System (6.x)

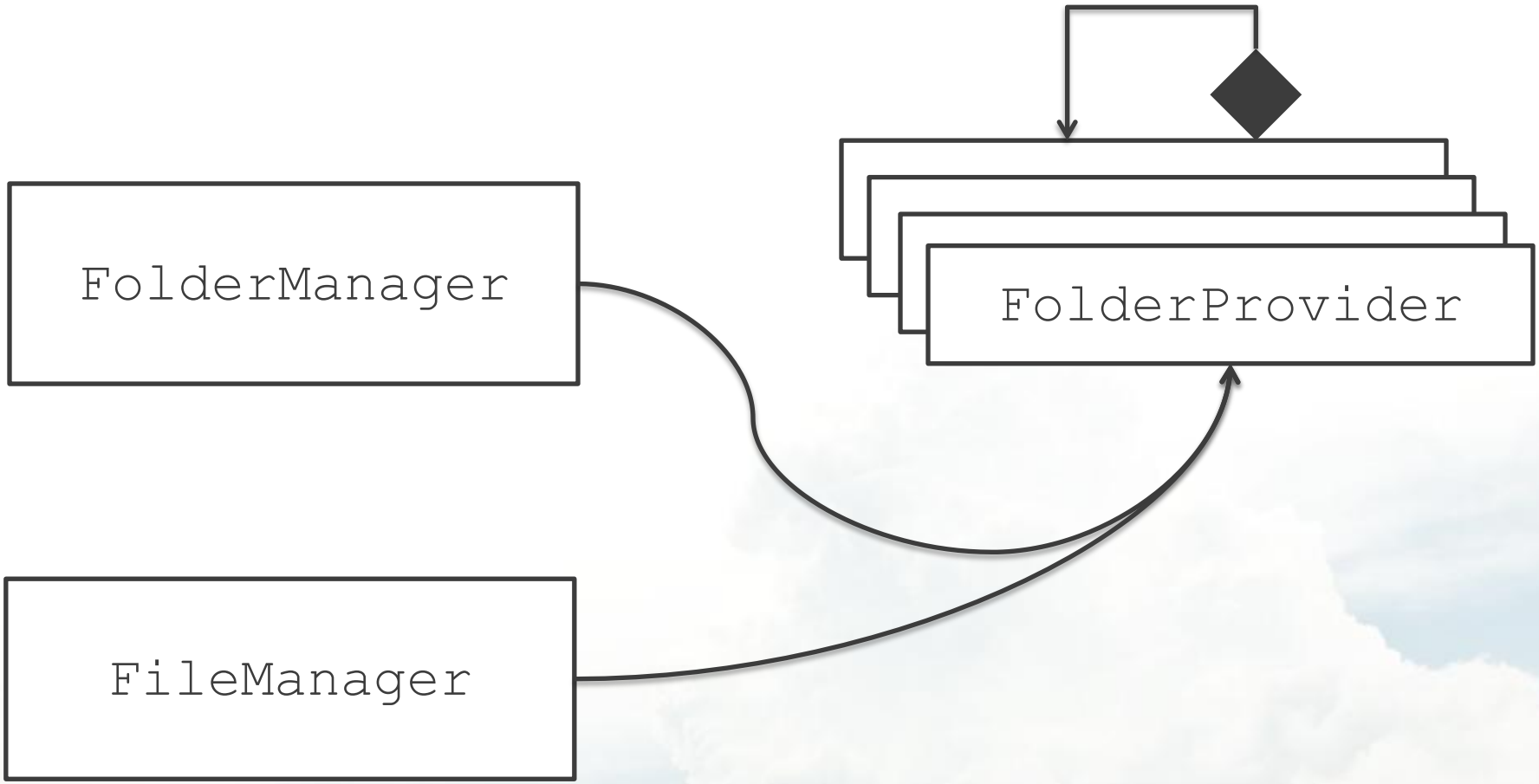



```
class FolderController
   {
   FolderInfo GetFolder(…)
      {
      return FolderManager
                  .Instance
                  .GetFolder(…);
      }
   }
```

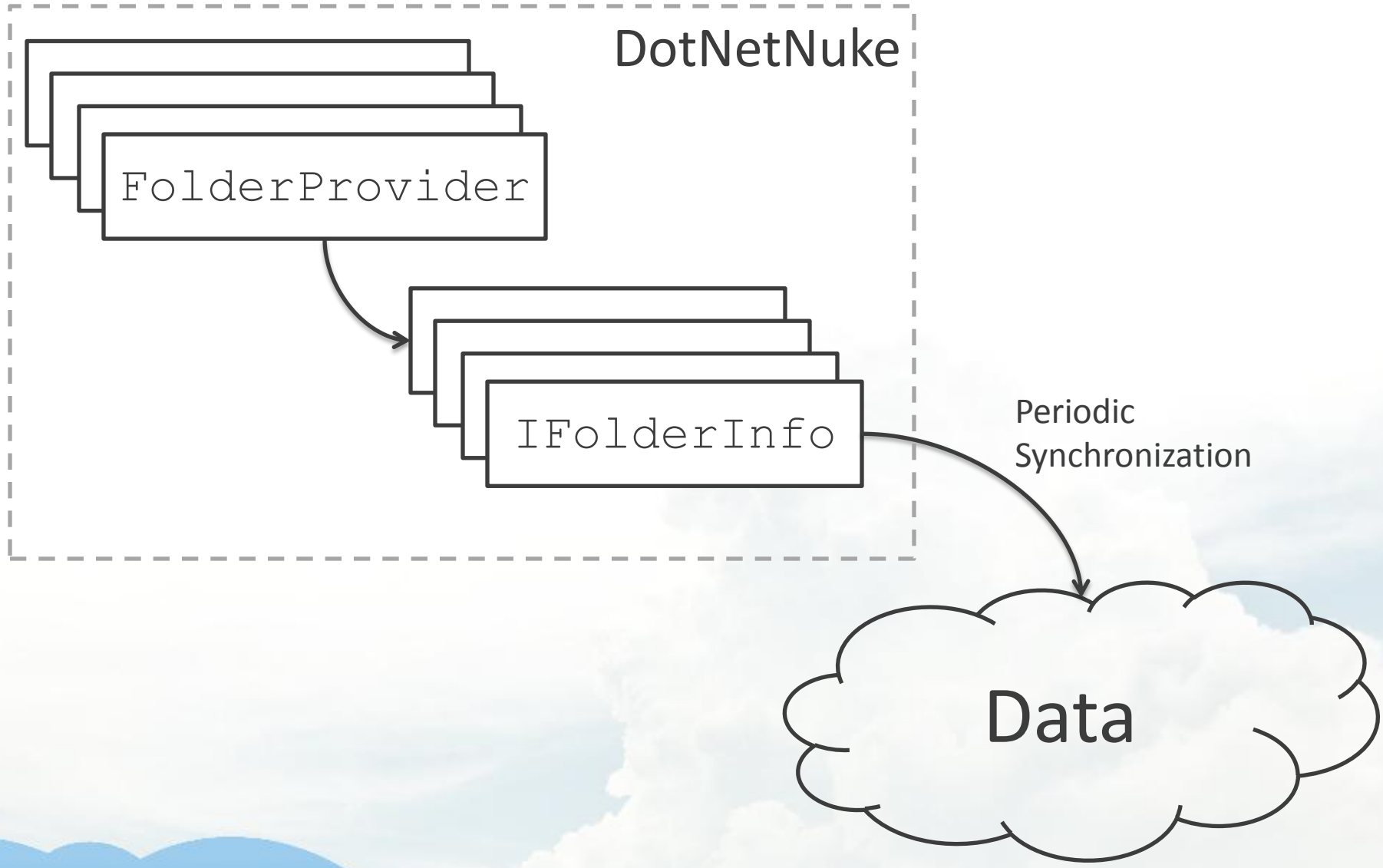
DotNetNuke File System (6.x)



DotNetNuke File System (6.x)



DotNetNuke File System (6.x)



web.config Provider Configuration

```
<[providerType]
   defaultProvider=“[name]">
  <providers>
   <clear/>
    <add
       name=“[name]"
       type=“[type]"
       otherAttributes=“…”/>
  </providers>
</[providerType]>
```

web.config Folder Provider Configuration

```
<folder
   defaultProvider="StandardFolderProvider">
  <providers>
   <clear/>
   <add name="StandardFolderProvider" .../>
   <add name="SecureFolderProvider" .../>
   <add name="DatabaseFolderProvider" .../>
</providers>
</roles>
```

web.config Folder Provider Configuration

```
<folder
   defaultProvider="StandardFolderProvider">
  <providers>
   <clear/>
   <add name="StandardFolderProvider" .../>
   <add name="SecureFolderProvider" .../>
   <add name="DatabaseFolderProvider" .../>
   <add name="MyAwesomeFolderProvider" .../>
</providers>
</roles>
```

FolderProvider Abstract Class

<<abstract>> FolderProvider	
IsStorageSecure RequiresNetworkConnectivity	
GetFiles GetFileStream GetFileSize GetFileUrl GetSubFolders AddFolder AddFile UpdateFile DeleteFolder DeleteFile RenameFile	RenameFolder FileExists FolderExists GetFileAttributes SetFileAttributes SupportsFileAttributes GetLastModificationTime GetSettingsVirtualControlPath IsInSync

FolderProvider Abstract Class

<<abstract>>
FolderProvider

```
GetFiles(IFolderInfo): string[]
GetFileStream(IFileInfo): Stream
GetFileSize(IFileInfo): long
GetFileUrl(IFileInfo): string
AddFile(folder: IFolderInfo, filename: string)
UpdateFile(file: IFileInfo, stream: Stream)
DeleteFile(IFileInfo)
RenameFile(file: IFileInfo, newName: string)
FileExists(file: IFolderInfo, name: string): Boolean
```

FolderProvider Abstract Class

<<abstract>>
FolderProvider

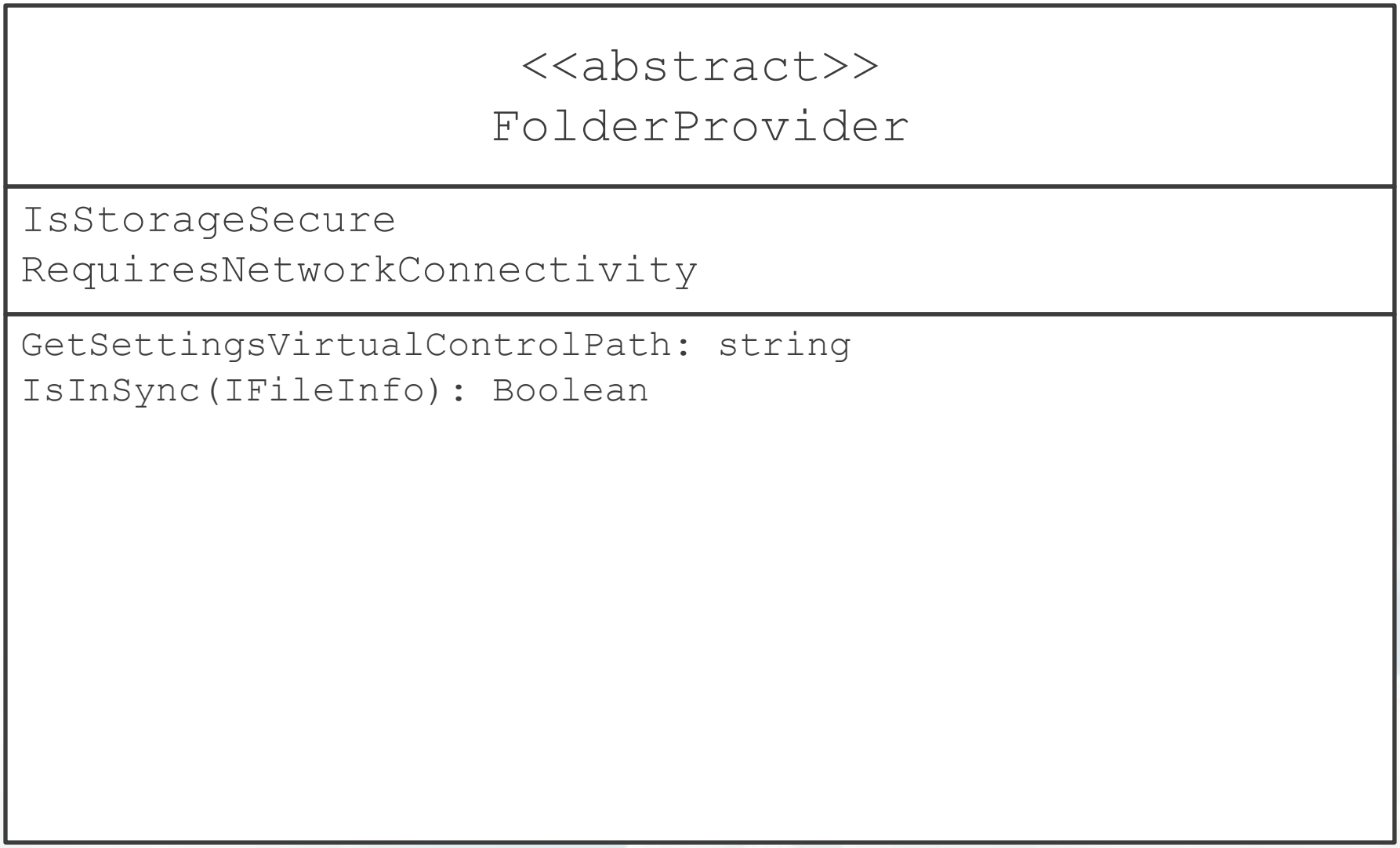
```
GetSubFolders(path: string, mapping: FolderMappingInfo)
AddFolder(path: string, mapping: FolderMappingInfo)
DeleteFolder(folder: IFolderInfo)
RenameFolder(folder: IFolderInfo, newName: string)
FolderExists(path: string, mapping: FolderMappingInfo)
```

FolderProvider Abstract Class

<<abstract>>
FolderProvider

```
GetFileAttributes(IFileInfo): System.IO.FileAttributes?
SetFileAttributes(IFileInfo, FileAttributes)
SupportsFileAttributes: Boolean
GetLastModificationTime(IFileInfo): DateTime
```

FolderProvider Abstract Class



Extension

File System Abstraction and Folder Providers in DotNetNuke

Custom File Providers

- Why would we need a custom file provider?
  - Probably going to use existing file system and cloud providers
    - E.g., existing Amazon S3 folder providers are “good enough”
    - New niches are likely to be quickly filled by third parties
- They’re more powerful than you think.
  - Not just for storing data on the cloud or in SharePoint
  - Essentially able to be a lightweight directory and stream service for arbitrary data
- Lots of possibilities:
  - Expose data via WCF, simple transformations, extend existing providers via decoration

Hello World

- Simplest provider possible
- Create a directory using the Hello-World provider
- Hello-World directories contain exactly one file:
  - `hello-world.txt`
- Not supported:
  - Sub-folders,
  - Uploads,
  - Updates,
  - Etc.

Symbolic Link File Provider

- Goals:
  - Create a symbolic link between child and parent portals
    - [PortalRoot]/MyFile.txt
      - http://folderprovider.local/Portals/0/MyFile.txt
    - [Child PortalRoot]/ParentPortal/MyFile.txt
      - http://folderprovider.local/Child/LinkClick.aspx
  - Parent files should be read-only in a child portal
  - Only link files with anonymous authorization
  - Synchronization, dialog selection, etc. should be natural and transparent

Symbolic Link Provider

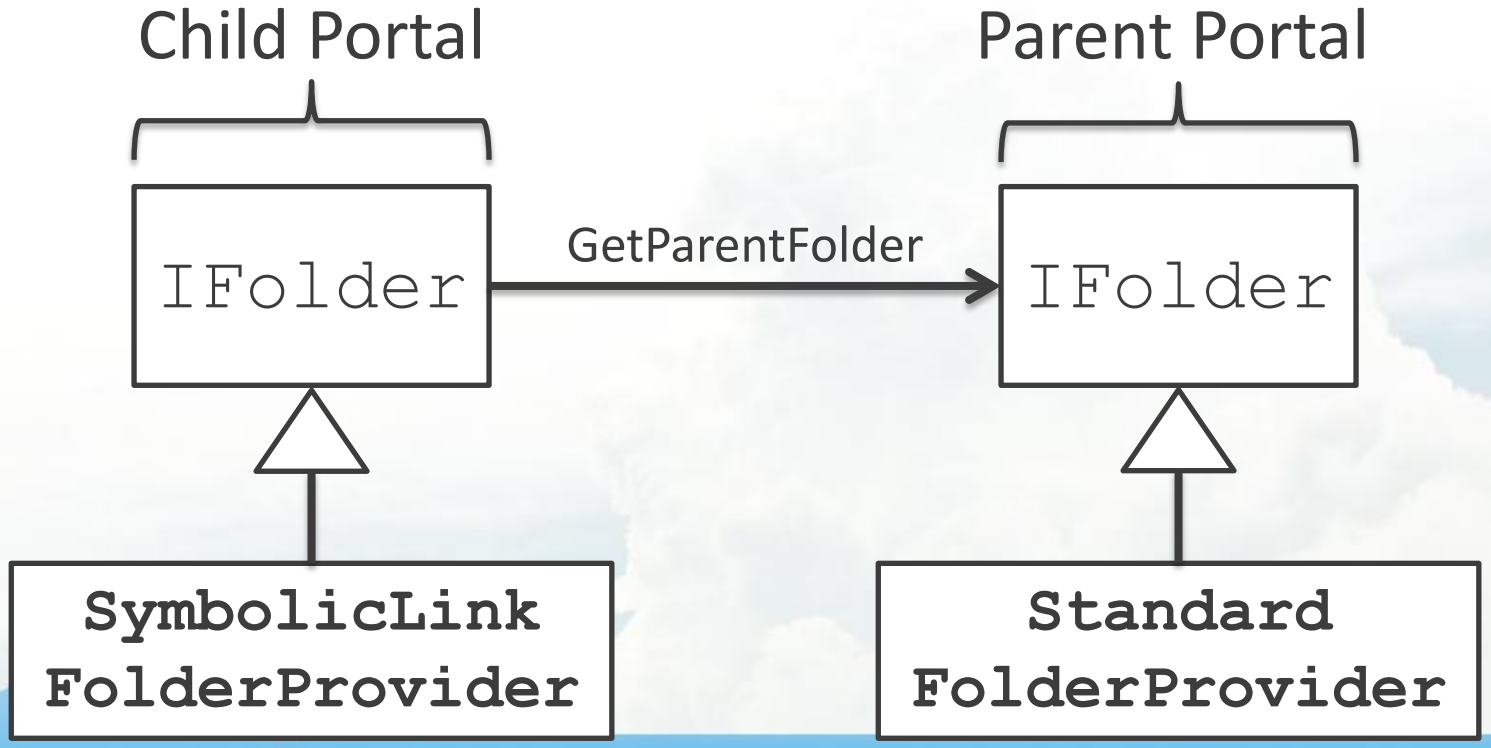
- Parent and Child Portal Aliases

http://folderprovider.local/Child

http://folderprovider.local

Symbolic Link Provider

- Mapping from child IFileInfo instances to parent IFileInfo instances
  - Child instance is of our custom provider type
  - Parent instance may be of any provider type!



Symbolic Link Provider

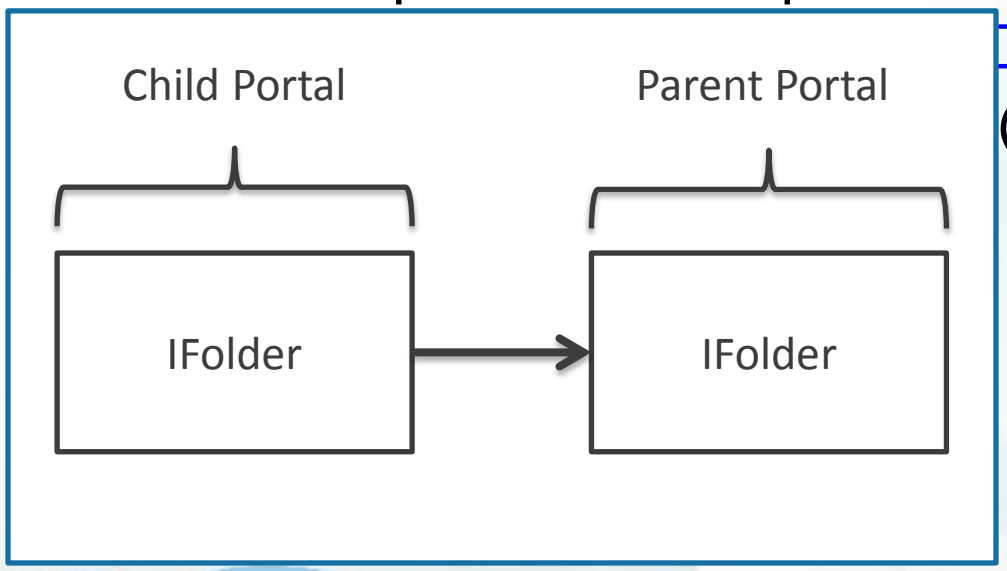
- Define an initial folder provider method:

```
override string[] GetFiles(IFolderInfo folder)
  {
  var parentFolder = folder.GetParentFolder();
  var provider = parentFolder.GetProvider();
  return provider != null
    ? provider.GetFiles(parentFolder)
    : new string[0];
  }
```

Symbolic Link Provider

- Define an initial folder provider method:

```
override string[] GetFiles(IFolderInfo folder)
  {
  var parentFolder = folder.GetParentFolder();
  var provider = parentFolder.GetProvider();
                                    ll
                                    (parentFolder)
```



Symbolic Link Provider

- Define an initial folder provider method:

```
override string[] GetFiles(IFolderInfo folder)
  {
  var parentFolder = folder.GetParentFolder();
  var provider = parentFolder.GetProvider();
  return provider != null
    ? provider.GetFiles(parentFolder)
    : new string[0];
  }
```

Symbolic Link Provider

- Define an initial folder provider method:

```
override string[] GetFiles(IFolderInfo folder)
  {
  var parentFolder = folder.GetParentFolder();
  var provider = parentFolder.GetProvider();
  return provider != null
    ? provider.GetFiles(parentFolder)
    : new string[0];
  }
```

Symbolic Link Provider

- Define an initial folder provider method:

```
override string[] GetFiles(IFolderInfo folder)
  {
Get Parent  var parentFolder = folder.GetParentFolder();
  var provider = parentFolder.GetProvider();
  return provider != null
    ? provider.GetFiles(parentFolder)
    : new string[0];
  }
```

Symbolic Link Provider

- Define an initial folder provider method:

```
override string[] GetFiles(IFolderInfo folder)
  {
  var parentFolder = folder.GetParentFolder();
  var provider = parentFolder.GetProvider();
  return provider != null
    ? provider.GetFiles(parentFolder)
    : new string[0];
  }
```

Get Provider

Symbolic Link Provider

- Define an initial folder provider method:

```
override string[] GetFiles(IFolderInfo folder)
  {
  var parentFolder = folder.GetParentFolder();
  var provider = parentFolder.GetProvider();
  return provider != null
    ? provider.GetFiles(parentFolder)
    : new string[0];
  }
```

Defer

Symbolic Link Provider

- Define an initial folder provider method:

```
override string GetFileUrl(IFileInfo file)
  {
  var parentFile = file.GetParentFile();
  var provider = parentFile.GetProvider();
  return provider != null
    ? provider.GetFileUrl(parentFile)
    : null;
  }
```

Symbolic Link Provider

- Define an initial folder provider method:

```
override string GetFileUrl(IFileInfo file)
  {
Get Parent  var parentFile = file.GetParentFile();
  var provider = parentFile.GetProvider();
  return provider != null
    ? provider.GetFileUrl(parentFile)
    : null;
  }
```

Symbolic Link Provider

- Define an initial folder provider method:

```
override string GetFileUrl(IFileInfo file)
  {
  var parentFile = file.GetParentFile();
  var provider = parentFile.GetProvider();
  return provider != null
    ? provider.GetFileUrl(parentFile)
    : null;
  }
```

Get Provider

Symbolic Link Provider

- Define an initial folder provider method:

```
override string GetFileUrl(IFileInfo file)
  {
  var parentFile = file.GetParentFile();
  var provider = parentFile.GetProvider();
  return provider != null
    ? provider.GetFileUrl(parentFile)
    : null;
  }
```

Defer

Symbolic Link Provider

- Define an initial folder provider method:

```
override string GetFileUrl(IFileInfo file)
  {
  var parentFile = file.GetParentFile();
  var provider = parentFile.GetProvider();
  return provider != null
    ? provider.GetFileUrl(parentFile)
    : null;
  }
```

Symbolic Link Provider

- Define an initial folder provider method:

```
override string GetFileUrl(IFileInfo file)
   {
   var parentFile = file.GetParentFile();
   var provider = parentFile.GetProvider();
   return provider != null
      ? provider.GetFileUrl(parentFile)
      : null;
   }
```

Symbolic Link Provider

- Even better, consider a helper function of type:

$$\begin{aligned} & \text{IFolderInfo} \\ & \longrightarrow (\text{FolderProvider} \longrightarrow (\text{IFolderInfo} \longrightarrow \alpha)) \\ & \longrightarrow \alpha \\ \longrightarrow & \ \alpha \end{aligned}$$

```
T ParentPortalOperation<T>(
  this IFolderInfo folder,
  Func<FolderProvider, Func<IFolderInfo, T>> operation,
  T defaultValue)
    {
    var parentFolder = folder.GetParentFolder();
    return parentFolder != null
      ? operation(parentFolder.GetProvider())(parentFolder)
      : defaultValue;
    }
```

Symbolic Link Provider

- Even better, consider a helper function of type:

IFolderInfo
⟶ (FolderProvider ⟶ (IFolderInfo ⟶ α))
⟶ α
⟶ α

```
T ParentPortalOperation<T>(
  this IFolderInfo folder,
  Func<FolderProvider, Func<IFolderInfo, T>> operation,
  T defaultValue)
    {
Get Parent  var parentFolder = folder.GetParentFolder();
    return parentFolder != null
      ? operation(parentFolder.GetProvider())(parentFolder)
      : defaultValue;
    }
```

Symbolic Link Provider

- Even better, consider a helper function of type:

IFolderInfo
⟶ (FolderProvider ⟶ (IFolderInfo ⟶ α))
⟶ α
⟶ α

```
T ParentPortalOperation<T>(
  this IFolderInfo folder,
  Func<FolderProvider, Func<IFolderInfo, T>> operation,
  T defaultValue)
    {
    var parentFolder = folder.GetParentFolder();
    return parentFolder != null
      ? operation(parentFolder.GetProvider())(parentFolder)
      : defaultValue;
    }
```

Get Provider

Symbolic Link Provider

- Even better, consider a helper function of type:

IFolderInfo
⟶ (FolderProvider ⟶ (IFolderInfo ⟶ α))
⟶ α
⟶ α

```
T ParentPortalOperation<T>(
  this IFolderInfo folder,
  Func<FolderProvider, Func<IFolderInfo, T>> operation,
  T defaultValue)
    {
    var parentFolder = folder.GetParentFolder();
    return parentFolder != null
Operation     ? operation(parentFolder.GetProvider())(parentFolder)
      : defaultValue;
    }
```

Symbolic Link Provider

- Initial folder provider method, revisited:

```
override string[] GetFiles(IFolderInfo folder)
  { return folder.ParentPortalOperation(
              provider => provider.GetFiles,
              new string[0]); }
```

- Other FolderProvider methods are similarly trivial:

```
override Stream GetFileStream(IFileInfo file)
  { return file.ParentPortalOperation<Stream>(
              provider => provider.GetFileStream); }
```

```
<component type="Config">
  <config>
    <configFile>web.config</configFile>
      <install>
        <configuration>
          <nodes>
            <node path="/configuration/dotnetnuke/folder/providers"
                  action="update" key="name" collision="overwrite">
              <add name="…"
                type="…" />
            </node>
          </nodes>
        </configuration>
      </install>
      <uninstall>
         <configuration>
           <nodes>
             <node path="…" action="remove" />
           </nodes>
         </configuration>
      </uninstall>
    </config>
</component>
```

Summary

- Folder providers are quite cool.
  - Think outside the box – not just for cloud file systems!
  - Function as a lightweight directory and data streaming service
  - Decorate standard providers for custom functionality
    - E.g., send an e-mail whenever a file is uploaded to a specific directory
- Modern code should interact with the file system via `FileManager.Instance` and `FolderManager.Instance`.
  - Move away from deprecated classes (e.g. `FileSystemUtilities`, `FileController`)
- New provider types need to derive from the `FolderProvider` abstract class
  - Only a handful of methods and you're up and running
  - Custom settings control via `GetSettingsVirtualControlPath`

2011 DotNetNuke WORLD ORLANDO

File System Abstraction and Folder Providers in DotNetNuke

Brandon Haynes