8

Creating a New Module

The next three chapters in this book will demonstrate various methods of customizing the CSK. Every community site will have different requirements to fulfill. Although the existing CSK framework offers a great deal of flexibility, having the entire source code available means you can add additional functionality to a site in an elegant manner. In this chapter, we will concentrate on creating a new module for the CSK. We will see how creating a new module allows you to add entirely new features which integrate seamlessly with the rest of the framework. In this chapter, we will implement a **Frequently Asked Questions** (**FAQ**) module.

Before we begin, let's mention one caveat. The CSK is a living piece of software. It will undoubtedly gain additional features and modules from the developer community, so one question you may want to answer is, "Has someone else already written the module I need?" Once you've made the commitment to customizing the CSK with your own code, you'll need to also think about integrating your code into newer versions of the CSK. If you stick to the current design used by the existing modules, chances are you'll find that the upgrades are easier.

Module Design

Before you begin implementing a new module for the CSK, you will first want to have a firm grasp of the features you wish to add, and then decide if any of the existing modules shipped with the CSK can fulfill that functionality.

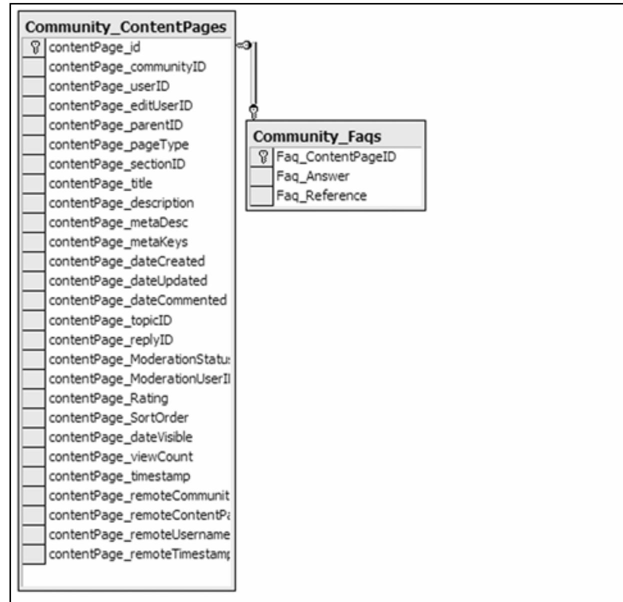
First, let us make a brief list of requirements for our community FAQ:

- An FAQ should consist of a question, an answer, a description or introduction, and pointers to additional references.
- Community users should have the ability to comment on and rank individual FAQs, as well as offer e-mail notifications when a new FAQ appears.
- Community users should have the ability to submit a new FAQ subject for the moderator's approval.

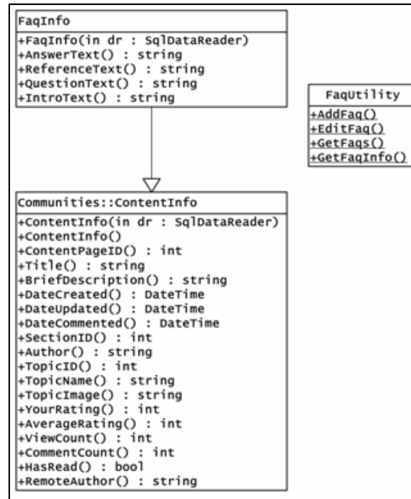
You could certainly create a list of questions and answers marked up in HTML and add the content to a community site using the HTML Page section type. However, the HTML Page section type offers limited user interaction (no comments, ranking, e-mails, or moderation).

Alternatively, the Articles section type could provide us with what we need, if we are willing to lump the FAQ answer and reference fields together in the article's body text. For maximum flexibility in presenting information, we would prefer to keep these as distinct entities. With our requirements and direction set, let's take a look at the classes and tables we will be building.

We know from the earlier chapters that the `Community_ContentPages` table will keep most of the information we need for an FAQ; for example, the author name, view count, and description. If we consider the question piece of the FAQ as the title, we really only need to store the FAQ answer and additional references as attributes. We will add a database table (`Community_Faqs`) as shown in the following diagram:

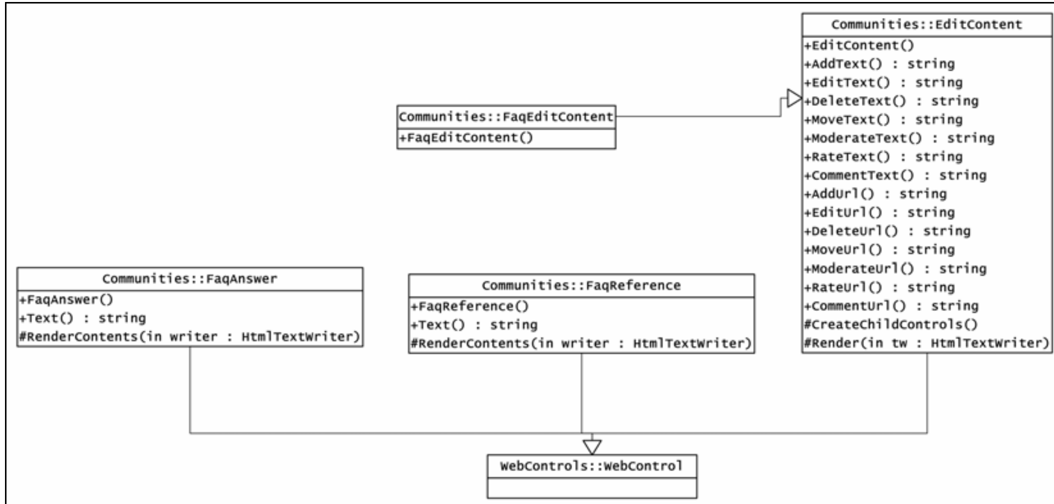


We can then build a class to hold FAQ information. As shown in the following diagram, the `FaqInfo` class inherits from `ContentInfo`, which holds most of the attributes for any content item. Every module also uses a utility class to retrieve, add, and edit content. For the FAQs module, this is the `FaqUtility` class (shown without method parameters).



We will also need to build classes for the code-behind pages that display and edit FAQs. We saw in earlier chapters how pages in the CSK derive from the `SkinnedCommunityControl` to allow themselves to be displayed with different skins. There are also a number of base classes with most of the behavior we need to add, edit, and display FAQs. The following diagram shows the class hierarchy that we will use for the FAQ code-behind classes:

We will also need to create `WebControl`-derived classes to display FAQ content. Typically, each attribute of our content will display in a distinct control, which allows a skin to lay out the content in whatever manner it sees fit. The following diagram shows the controls that we will use for this module, all of which ultimately derive from `WebControl`:



Module Construction Blueprint

We will take a bottom-up approach by starting with the database work, and finishing with presentation skins and themes. We will follow the naming conventions and patterns established by the existing modules in the CSK so that the code fits well with the rest of the framework. For example, the Books module retrieves book information from the `Community_Books` table to populate a `BookInfo` component.

Thus we will use a table called `Community_Faqs` to populate an `FaqInfo` component. However, you may want to consider adding a unique identifier in case a future version of the CSK contains the module you are building. For instance, if you work for ABC Inc. you might use `Community_ABCFaqs` as a table name to lower the possibility of future name collisions.

We will use the following steps to build the FAQ module. You can also follow these steps in a general sense to build your own modules:

1. Create a new table (`Community_Faqs`) to hold the additional fields for the new module.
2. Create stored procedures to add, edit, and select a single FAQ, and a stored procedure to select all FAQs for a given section.
3. Create a maintenance stored procedure to initialize the FAQ module by populating the `Community_PageTypes` and `Community_NamedPages` tables.
4. Create a component (`FaqInfo`) to hold information on a single FAQ.

5. Create a component (FaqUtility) with data-access routines to invoke our FAQ-related stored procedures.
6. Create controls derived from WebControl to display the individual module fields. These controls will be FaqQuestion, FaqIntro, FaqAnswer, FaqReference, and FaqEditContent.
7. Create controls derived from the SkinnedCommunityControl class to contain the logic behind the page content skins from the next step. These controls will be AddFaq, EditFaq, FaqSection, and Faq.
8. Create new page-content skins for the module. This includes Faqs_AddFaq.ascx, Faqs_FaqSection.ascx, and Faq_Faq.ascx. We will use the Faqs_AddFaq skin to add a new FAQ as well as edit an existing FAQ. At a minimum you will need to create a default skin in the ContentSkins folder under the Communities\Common\Themes\Default\Skins directory. You can optionally create additional skins for other available themes, such as the Robotico and Professional themes.
9. Create style rules in the CSS files in Communities\Common\Themes\Default\Styles for the module. You should also add CSS rules into all of the CSS files in the themes your community may use.

The rest of this chapter will explain each step in more detail.

The Community_Faq Table

Content that is common to all modules such as title, description, and view count resides in the Community_ContentPages table. Additional module-specific content requires a second table for storage. For the FAQ module, we can store the FAQ question in the contentPage_title field of Community_ContentPages, and the FAQ introduction in the contentPage_description field. We still need to store the FAQ answer and the additional references text for the FAQ, so we will use the following DDL to create a table:

```
CREATE TABLE [Community_Faqs] (
    [Faq_ContentPageID] [int] NOT NULL ,
    [Faq_Answer] [ntext] NOT NULL ,
    [Faq_Reference] [ntext] NULL,
    CONSTRAINT [PK_Community_Faqs] PRIMARY KEY CLUSTERED
    (
        [Faq_ContentPageID]
    ),
    CONSTRAINT [FK_Community_Faqs_Community_ContentPages]
           FOREIGN KEY
    (
        [Faq_ContentPageID]
    ) REFERENCES [Community_ContentPages] (
        [contentPage_id]
    ) ON DELETE CASCADE
)
```

The naming conventions and data types we use for the table are consistent with the other modules shipped with the CSK.

We store the answer and reference data in fields of type ntext in order to support large quantities of text (up to 1 GB). Also notice how Faq_Answer is a required field but Faq_Reference may contain a NULL value. Our primary key (Faq_ContentPageID) references the additional FAQ content in the Community_ContentPages table. The foreign key constraint will ensure our referential integrity for this relationship.

Another nice design feature is the use of cascading referential integrity restraints. The ON DELETE CASCADE clause in the foreign key constraint means we will not need to write any code to delete an FAQ record from Community_Faqs.

The CSK uses the Community_ContentPagesDeleteContentPage stored procedure to delete records from the Community_ContentPages table. When the procedure removes a record from the content table, SQL Server will automatically remove the corresponding entry from the FAQ table.

The Community_Faqs Stored Procedures

The next steps in our blueprint calls for us to create stored procedures to add an FAQ, edit an FAQ, retrieve a single FAQ, and retrieve a paged and sorted list of FAQs, for a total of four procedures. All of the data access in the CSK happens via stored procedures. There is no ad hoc SQL in the code, which is a good practice from both encapsulation and security standpoints. The first procedure we will write is to add new FAQ content.

Community_FaqsAddFaq

The stored procedure for adding a new FAQ to the database is shown now. We do not need to provide a parameter for every column in the two content tables. For example, we do not need to pass a parameter to populate the contentPage_ViewCount column. Many of the columns contain a sensible default value (contentPage_ViewCount defaults to 0) or allow NULL values (contentPage_dateCommented) for new content.

```
CREATE PROCEDURE Community_FaqsAddFaq
(
  @communityID int,
  @sectionID int,
  @username nvarchar(50),
  @topicID int,
  @question nvarchar(100),
  @introduction nvarchar(500),
  @metaDescription nvarchar(250),
  @metaKeys nvarchar(250),
  @moderationStatus int,
  @answer ntext,
  @reference ntext
)
AS
DECLARE @ContentPageID int
DECLARE @pageType int
SET @pageType = dbo.Community_GetPageTypeFromName('Faq')
DECLARE @userID int
SET @userID = dbo.Community_GetUserID(@communityID, @username);
BEGIN TRAN
EXEC @ContentPageID = Community_AddContentPage
  @communityID,
  @sectionID,
  @userID,
```

```
    @question,
    @introduction,
    @metaDescription,
    @metaKeys,
    @pageType,
    @moderationStatus,
    @topicID

  INSERT Community_Faqs
  (
    Faq_ContentPageID,
    Faq_Answer,
    Faq_Reference
  )
  VALUES
  (
    @ContentPageID,
    @answer,
    @reference
  )
  COMMIT TRAN
  RETURN @ContentPageID
```

Notice the use of two UDFs supplied with the CSK. The first UDF retrieves the page type for the content. There is a distinct page type identifier for each module (Books, Articles, Downloads, and so on) in the CSK. We will discuss page types in more detail when we create our maintenance stored procedure. A second UDF allows us to retrieve the `userID` by passing a `communityID` and `username`.

Since we must insert the content items into two different tables we use a transaction to make the operation atomic. Inserting records into the `Community_ContentPages` table occurs by invoking the `Community_AddContentPage` procedure passing the FAQ question as the `@Title` parameter and the FAQ introduction as the `@Description` parameter. `Community_AddContentPage` returns the primary key value of the newly inserted record which we will in turn use in the `INSERT` command for `Community_Faqs`.

> All of the procedures that add new content records must return the primary key value of the new record as a result.

The new `contentPageID` value will be useful in the upper layers of software, as we will see when we write the data-access component.

Community_FaqsEditFaq

The stored procedure we use to edit an existing FAQ uses a slightly different parameter list. Obviously, some columns are immutable after we add a content page to the system (such as the section identifier). The edit procedure listing is shown below:

```
  CREATE PROCEDURE Community_FaqsEditFaq
  (
    @communityID int,
    @contentPageID int,
    @username NVarchar(50),
    @topicID int,
    @question NVarchar(100),
    @introduction NVarchar(500),
```

```
  @metaDescription NVarchar(250),
  @metaKeys NVarchar(250),
  @answer Text,
  @reference Text
)
AS
DECLARE @UserID int
SET @UserID = dbo.Community_GetUserID(@communityID, @username)
EXEC Community_EditContentPage
    @contentPageID,
    @userID,
    @question,
    @introduction,
    @metaDescription,
    @metaKeys,
    @topicID
UPDATE Community_Faqs SET
  Faq_Answer = @answer,
  Faq_Reference = @reference
WHERE Faq_ContentPageID = @contentPageID
```

Again, we use a stored procedure provided by the CSK to update the `Community_ContentPages` pages, and then follow with an `UPDATE` statement for the `Community_Faqs` table. Unlike the procedure to add a new FAQ record, there is no transaction present to keep the two table updates atomic. We are following a pattern established in the existing module edit procedures in the CSK—none of these use a transaction. Presumably the designers found the consequences of a failure during a content edit to be considerably smaller compared to the same during content addition. We should have slightly better system throughput by avoiding SQL resource locks.

Community_FaqsGetFaqs

The next stored procedure to write is the procedure to return all FAQs for a given section in a community. The `Community_GetPagedSortedContent` UDF, which we covered earlier in the book, should essentially dictate the parameter list required to retrieve content. All we need to do is augment the resultset of the UDF with some FAQ-specific columns and sort by the `IndexID` column the UDF computes.

```
CREATE PROCEDURE Community_FaqsGetFaqs
(
  @communityID int,
  @username NVarchar(50),
  @sectionID int,
  @pageSize int,
  @pageIndex int,
  @sortOrder NVarchar(50)
)
AS
DECLARE @currentDate DATETIME
SET @currentDate = GetUtcDate()
SELECT
    null Faq_Answer,
    null Faq_Reference,
    Content.*
FROM
    dbo.Community_GetPagedSortedContent
    (
        @communityID,
        @username,
        @sectionID,
        @currentDate,
```

```
            @sortOrder,
            @pageSize,
            @pageIndex,
            default
        ) Content
    ORDER BY
        IndexID
```

The stored procedure uses a couple of techniques to reduce the amount of code we need to write and to reduce the amount of maintenance required in future changes. First, we use `Content.*` in the `SELECT` list to return all columns in the UDF resultset. This code is consistent with the other stored procedures within the CSK. In terms of efficiency, it is better to explicitly list all the columns required instead of having the database engine figure out the available columns. However, in this case, the CSK designers decided to lean towards maintainability. With this code you can make certain types of schema changes to the `Community_ContentPages` (adding a column, for example) and not need to modify and test any of the stored procedures (almost 30) working with records from this table.

The second thing to point out is the addition of two empty columns (`Faq_Answer` and `Faq_Reference`) to the resultset. Later, we will write the `FaqInfo` component to hold results of both this stored procedure *and* the next stored procedure that retrieves a single FAQ. Since we want to use the same component for both operations, we need to populate the resultset with all the columns an `FaqInfo` object expects to see. As these two columns can be quite large, and would never display in a summary list of FAQs, we do not want to use up resources moving these columns around for the FAQ list, we just set the values to `NULL`.

Community_FaqsGetFaq

The stored procedure we write to retrieve the content for a single FAQ also has one other responsibility. It needs to increment the view count for the content page and indicate the user has now read this content page. Both these tasks are accomplished by executing the CSK's `Community_ContentPagesTrackStats` procedure. The entire procedure is shown here:

```
    CREATE PROCEDURE Community_FaqsGetFaq
    (
      @communityID INT,
      @username NVarchar(50),
      @contentPageID int
    )
    AS
    DECLARE @userID INT
    SET @userID = dbo.Community_GetUserID(@communityID, @username)
    -- Update ViewCount and HasRead Stats
    EXEC Community_ContentPagesTrackStats @userID, @contentPageID
    DECLARE @currentDate DATETIME
    SET @currentDate = GetUtcDate()
    SELECT
        Faq_Answer,
        Faq_Reference,
        Content.*
    FROM
        dbo.Community_GetContentItem(
            @communityID,
            @userID,
            @currentDate) Content
```

```
    JOIN Community_Faqs (nolock)
        ON ContentPage_ID = Faq_ContentPageID
WHERE
    ContentPage_ID = @contentPageID
```

Note that this time we actually retrieve the value for the `Faq_Answer` and `Faq_Reference` columns, as they should display at this level of detail. This requires a join to our `Community_Faqs` table. You'll see that we include a locking hint of `nolock`, which allows dirty reads but avoids any contention for the content we retrieve.

Initializing the FAQ Module

Every community module has a corresponding maintenance stored procedure to populate the database with settings required for the module to work. Specifically, we need to register content page types by inserting two records into `Community_PageTypes`: one for an FAQ section page (to display a list of FAQs) and one for an FAQ page (showing a single FAQ in detail). We call the procedure `Community_MaintenanceInitializeFaqs`, following the existing CSK naming convention. An excerpt registering the FAQ section page type is shown here:

```
IF NOT EXISTS (SELECT * FROM Community_PageTypes WHERE pageType_Name='Faq
Section')
    BEGIN
    INSERT Community_PageTypes
    (
      pageType_name,
      pageType_description,
      pageType_pageContent,
      pageType_IsSectionType,
      pageType_ServiceSelect
    )
    VALUES
    (
      'FAQ Section',
      'Contains FAQs in a question and answer style format',
      'ASPNET.StarterKit.Communities.Faqs.FaqSection',
      1,
      'Community_FaqsServiceSelect'
    )
    END
ELSE
    PRINT 'WARNING: The FAQ Module has already been registered.'
```

> The CSK caches data from `Community_NamePages` so as to retrieve the data only once. If you make modifications to the table, you'll need to restart the web application for the changes to take effect in the CSK.

The maintenance stored procedure also needs to register the named pages (static content) for the new module. Named pages for the FAQ section will include the page to add an FAQ and a page to edit an FAQ. You'll have to choose your page names at this point and use the same names later when you create the ASPX file.

Here's an excerpt from Community_MaintenanceInitializeFaqs to add a named page for adding FAQs:

```
IF NOT EXISTS (SELECT * FROM Community_NamedPages WHERE
namedPage_Path='/Faqs_AddFaq.aspx')
    BEGIN
    INSERT Community_NamedPages
    (
      namedPage_name,
      namedPage_path,
      namedPage_pageContent,
      namedPage_title,
      namedPage_description,
      namedPage_sortOrder,
      namedPage_isVisible,
      namedPage_menuID
    )
    VALUES
    (
      'AddFaq',
      '/Faqs_AddFaq.aspx',
      'ASPNET.StarterKit.Communities.Faqs.AddFaq',
      'Add FAQ',
      'Enables users to add a new FAQ',
      0,
      1,
      0
    )
    END
ELSE
    PRINT 'WARNING: /Faqs_AddFaq.aspx has already been registered as a
NamedPage.'
```

The namedPage_pageContent parameter is the name of the class that the CSK will instantiate as the code-behind logic for the page. The name includes the full namespace qualifier ASPNET.StarterKit.Communities.Faqs.AddFaq.

The maintenance stored procedure needs to execute during the database setup. We will take a look at how to do this in Chapter 11.

FAQ Components

The C# code for our FAQ module will reside in the Engine\Modules\Faqs directory. First, we will write out helper components and place these in a Components directory. Each module in the CSK places components inside a distinct namespace below ASPNET.StarterKit.Communities, and the existing modules use the name of the module as the additional namespace qualifier (Faqs).

FaqInfo

FaqInfo class extends the ContentInfo class to offer data properties specific to an FAQ. The code for this class is shown as follows:

```
using System;
using System.Data.SqlClient;
namespace ASPNET.StarterKit.Communities.Faqs
{
```

```
        public class FaqInfo : ContentInfo
        {
            public FaqInfo(SqlDataReader dr) : base(dr)
            {
                if(dr["Faq_Answer"] != DBNull.Value)
                {
                    _answerText = (string)dr["Faq_Answer"];
                }
                if(dr["Faq_Reference"] != DBNull.Value)
                {
                    _referenceText = (string)dr["Faq_Reference"];
                }
            }
            public string AnswerText
            {
                get { return _answerText; }
                set { _answerText = value; }
            }
            public string ReferenceText
            {
                get { return _referenceText; }
                set { _referenceText = value; }
            }
            public string QuestionText
            {
                get { return base.Title; }
                set { base.Title = value; }
            }
            public string IntroText
            {
                get { return base.BriefDescription; }
                set { base.BriefDescription = value; }
            }
            private string _answerText;
            private string _referenceText;
        }
    }
```

FaqInfo expects initialization with an instance of the SqlDataReader class. We will be writing the data-access code to create a SqlDataReader in our next class.

FaqUtility

Following the patterns set forth in the rest of the CSK, we will put all of our data-access routines into static methods of a utility class. There should be one static method available for each of the FAQ-related stored procedures (with the exception of the maintenance stored procedure, which we should not need to invoke during regular operations of the community site but only during setup). Each of these routines will need to map incoming variables to stored procedure parameters and execute the procedure.

Here's the AddFaq method:

```
    public static int AddFaq(
                        string username,
                        int sectionID,
                        int topicID,
                        string question,
                        string introduction,
                        string answer,
                        string reference,
                        int moderationStatus)
    {
```

```
        SqlConnection conPortal = new SqlConnection(
                    CommunityGlobals.ConnectionString);
        SqlCommand cmdAdd = new SqlCommand(
                    "Community_FaqsAddFaq", conPortal);
        cmdAdd.CommandType = CommandType.StoredProcedure;
        cmdAdd.Parameters.Add("@RETURN_VALUE",
                              SqlDbType.Int).Direction =
                               ParameterDirection.ReturnValue;
        cmdAdd.Parameters.Add("@communityID",
                                CommunityGlobals.CommunityID);
        cmdAdd.Parameters.Add("@sectionID", sectionID);
        cmdAdd.Parameters.Add("@username", username);
        cmdAdd.Parameters.Add("@topicID", topicID);
        cmdAdd.Parameters.Add("@question", question);
        cmdAdd.Parameters.Add("@introduction", introduction);
        cmdAdd.Parameters.Add("@metaDescription",
            ContentPageUtility.CalculateMetaDescription(introduction));
        cmdAdd.Parameters.Add("@metaKeys",
            ContentPageUtility.CalculateMetaKeys(introduction));
        cmdAdd.Parameters.Add("@moderationStatus", moderationStatus );
        cmdAdd.Parameters.Add("@answer", SqlDbType.NText);
        cmdAdd.Parameters.Add("@reference", SqlDbType.NText);
        cmdAdd.Parameters["@answer"].Value = answer;
        cmdAdd.Parameters["@reference"].Value = reference;

        conPortal.Open();
        cmdAdd.ExecuteNonQuery();
        int result = (int)cmdAdd.Parameters["@RETURN_VALUE"].Value;
        SearchUtility.AddSearchKeys(conPortal, sectionID, result,
                                    question, introduction);
        conPortal.Close();
        return result;
    }
```

Notice that the `AddFaq` method also generates the search keys for the content using the `SearchUtility` class, and the newly created identifier of the content returned by the stored procedure we reviewed earlier.

The `EditFaq` method almost duplicates the `AddFaq` method except for calling a different stored procedure and using `EditSearchKeys` on the `SearchUtility` class to update the FAQ search keys.

One improvement you might consider making to the CSK is adding a `try catch finally` statement to ensure the database connection will always invoke the `Close` method, even in the face of an exception. The chances of an exception are small, but on a high volume community site, you cannot afford the opportunity to waste database connections.

The other two methods in `FaqUtility` are `GetFaqs` and `GetFaqInfo`. `GetFaqs` loops through records in a `SqlDataReader` to return an ArrayList of `FaqInfo` objects, while `GetFaqInfo` expects only a single record in the database results and returns a single new `FaqInfo` object. These two methods from the class are shown here:

```
    public static ContentInfo GetFaqInfo(string username, int contentPageID)
    {
        FaqInfo faq = null;
        SqlConnection conPortal = new SqlConnection(
                        CommunityGlobals.ConnectionString);
        SqlCommand cmdGet = new SqlCommand(
                        "Community_FaqsGetFaq", conPortal);
        cmdGet.CommandType = CommandType.StoredProcedure;
        cmdGet.Parameters.Add(
                        "@communityID", CommunityGlobals.CommunityID);
        cmdGet.Parameters.Add("@username", username);
        cmdGet.Parameters.Add("@contentPageID", contentPageID);
```

```
        conPortal.Open();
        SqlDataReader dr = cmdGet.ExecuteReader();
        if (dr.Read())
            faq = new FaqInfo(dr);
        conPortal.Close();
        return faq;
    }
    public static ArrayList GetFaqs(string username, int sectionID,
                          int pageSize, int pageIndex, string sortOrder)
    {
        SqlConnection conPortal = new
    SqlConnection(CommunityGlobals.ConnectionString);
        SqlCommand cmdGet = new SqlCommand( "Community_FaqsGetFaqs",
    conPortal);
        cmdGet.CommandType = CommandType.StoredProcedure;
        cmdGet.Parameters.Add("@communityID",
                        CommunityGlobals.CommunityID);
        cmdGet.Parameters.Add("@username", username);
        cmdGet.Parameters.Add("@sectionID", sectionID);
        cmdGet.Parameters.Add("@pageSize", pageSize);
        cmdGet.Parameters.Add("@pageIndex", pageIndex);
        cmdGet.Parameters.Add("@sortOrder", sortOrder);

        ArrayList faqs = new ArrayList();
        conPortal.Open();
        SqlDataReader dr = cmdGet.ExecuteReader();
        while (dr.Read())
            faqs.Add(new FaqInfo(dr));
        conPortal.Close();
        return faqs;
    }
```

It is important for `GetFaqInfo` to use the return value and parameter list shown above. The framework should invoke these methods through a delegate and the signatures must match. We will see how this works when we write the content pages.

Our data-access layer is now complete. If you build a module in this fashion, you should be able to compile the solution at this time to resolve any errors. You might consider writing a driver page to exercise the four static methods in `FaqUtility` and verify the results by looking in the `Community_Faqs` and `Community_ContentPages` tables of the database.

FAQ WebControls

The CSK breaks up the display of content into smaller controls. For example, under `Engine\Framework\ContentPages\Controls`, you'll find a control to display the content title (in `Title.cs`), and the content's brief description (`BriefDescription.cs`), which can display our FAQ question and the introduction. All we will need to add are a couple of controls specific to the FAQ module: a control to display the answer and the reference, and a control to provide a link for authorized users to edit the FAQ content.

FaqAnswer and FaqReference

All of the controls at this level derive from the .NET Framework `WebControl` class. We simply need to set the `CssClass` property for our control, retrieve the text to display from the current `HttpContext`, and override the `RenderContents` method to write the text.

Create these controls in the Engine\Module\Faqs\Controls directory. The control to display the answer to an FAQ is as follows:

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using ASPNET.StarterKit.Communities.Faqs;
using System.ComponentModel;
namespace ASPNET.StarterKit.Communities
{
    [Designer(typeof(ASPNET.StarterKit.Communities.CommunityDesigner))]
    public class FaqAnswer : WebControl
    {
        public FaqAnswer() : base()
        {
            CssClass = "faqAnswerText";
            if(Context != null)
            {
                Object faqInfo = Context.Items["ContentInfo"];
                if(faqInfo != null)
                {
                    _text = ((FaqInfo)faqInfo).AnswerText;
                }
            }
        }
        public string Text
        {
            get { return _text; }
            set { _text = value; }
        }
        override protected void RenderContents(
                         HtmlTextWriter writer)
        {
            SectionInfo objSectionInfo =
                         (SectionInfo)Context.Items["SectionInfo"];
            writer.Write(
                   CommunityGlobals.FormatText(
                                objSectionInfo.AllowHtmlInput,
                                objSectionInfo.ID, _text));
        }
        private string _text;
    }
}
```

First, notice that our previous components were in the ASPNET.StarterKit.Communities.Faqs namespace, but we place our control in a higher level namespace. This remains consistent with the rest of the CSK where the skin files typically set up a Community tag prefix pointing to the ASPNET.StarterKit.Communities namespace. We can place the above component into a skin with a line similar to:

```
<community:FaqAnswer Runat="Server" ID="Answer1"
  NAME="Answer1"/>
```

The FaqReference control looks very similar to the FaqAnswer control. Both override RenderContents and use the CommunityGlobals class for proper transformation and formatting of the outgoing text. Generally, you will want to break each field of a new module into a specific control to use from a content skin. It is up to the skin designer to decide where to lay out the fields for presentation.

FaqEditContent

Every CSK module uses a control derived from `EditContent` to display links for authorized users to add, delete, move, comment, and moderate content. All we need to do is set the appropriate URL properties. The logic in the base class will determine when to display the appropriate links. We perform all of the work in the constructor, as follows:

```
public FaqEditContent()
{
    if (Context != null)
    {
        PageInfo pageInfo = (PageInfo)Context.Items["PageInfo"];
        int contentPageID = pageInfo.ID;
        AddUrl = "Faqs_AddFaq.aspx";
        EditUrl = String.Format(
                        "Faqs_EditFaq.aspx?id={0}",
                        contentPageID);
        DeleteUrl = String.Format(
                        "ContentPages_DeleteContentPage.aspx?id={0}",
                        contentPageID);
        MoveUrl = String.Format(
                        "ContentPages_MoveContentPage.aspx?id={0}",
                        contentPageID);
        CommentUrl = String.Format(
                        "Comments_AddComment.aspx?id={0}",
                        contentPageID);
        ModerateUrl = "Moderation_ModerateSection.aspx";
    }
}
```

With these small pieces built we are ready to tackle the actual content display.

Content Classes

In the traditional ASP.NET paradigm, the content classes are the *code-behind* files. Since the CSK takes a slightly different tack to allow high levels of customization, we will not be able to use the IDE to keep our web form in sync with the code behind the form. No real tie exists between the two, since each code file supports multiple versions of the same web form (the *skins*). Instead, we will need to manually keep track of which controls are on the page, and manually wire up the events that we need. The task is not difficult but does require some extra attention to control names and other details.

We have four content classes to write for the four pages we will need for our FAQ module:

- `Faq`: To display a single FAQ item
- `FaqSection`: To display a list of FAQs
- `AddFaq`: For the data entry of FAQ content
- `EditFaq`: For the update of FAQ content

The amount of code you will need to write for a content class varies widely. Using the `ContentItemPage` and `ContentListPage` CSK classes we can display an FAQ and an FAQ list with very little code. We will look at these two classes first.

Faq and FaqSection

The FaqSection class derives from ContentListPage, which can do most of our work with just a little more of information.

```
public class FaqSection : ContentListPage
{
    public FaqSection() : base()
    {
        SkinFileName = _skinFileName;
        GetContentItems = _getContentItems;
    }
    string _skinFileName = "Faqs_FaqSection.ascx";
    GetContentItemsDelegate _getContentItems =
              new GetContentItemsDelegate(FaqUtility.GetFaqs);
}
```

At this point, we need to pick the actual filename for our skin and set the SkinFileName property. This step is essential to SkinnedCommunityControl (the base class of ContentListPage) for finding the correct skin to load.

When we wrote the data-access methods in FaqUtility, we mentioned the need to maintain a specific method signature of return type and parameters. The method signature we used was the one for a GetContentItemsDelegate delegate. The base class will use the delegate within the BindContent method to retrieve and display all the FAQs in a given section. The Faq class follows the same pattern, but initializes the skin file to Faqs_Faq.ascx, and assigns the delegate to the FaqUtility.GetFaq method.

AddFaq and EditFaq

These two classes present a bit of a challenge. Since the data-access methods to insert and update module content vary widely from module to module, there is no base class available to reduce the workload through a delegate. Instead, we will need to find controls that are specific to our module to get and set values, and invoke the FaqUtility data-access routines in response to user events.

Before reaching this point, you may want to sketch out a skin file to know what controls you will need on the page. We know the skin file that we will use to add and edit FAQs will need the following input controls:

- TextBox: For the FAQ question
- TextBox: For the FAQ introduction
- TopicPicker: For FAQ sections supporting topics
- HtmlTextBox: For the FAQ answer
- HtmlTextBox: For the FAQ references

In addition, we would like to preview the control, which requires five more controls for display instead of input. These five controls should be the same as the ones we will use in the display of an FAQ. So we will use the FaqAnswer control we wrote earlier to display the answer. Let us look at the EditFaq class as an example.

The constructor is as follows:

```
public EditFaq() : base()
{
  SkinFileName = _skinFileName;
  SectionContent = _sectionContent;

  this.SkinLoad += new SkinLoadEventHandler(SkinLoadFaq);
  this.Preview += new PreviewEventHandler(PreviewFaq);
  this.Submit += new SubmitEventHandler(SubmitFaq);
}
```

The constructor initializes the skin file name and section content properties, which we will define later. The constructor then wires up event handlers for three events defined in the base class `ContentEditPage`. These event handlers will contain the logic for loading the skin, handling the preview button click, and the submit button click. They are a part of every content edit page.

```
void SkinLoadFaq(Object s, SkinLoadEventArgs e)
{
  txtQuestion = (TextBox)GetControl(e.Skin, "txtQuestion");

      // continue initializing all controls with GetControl . . .
}
```

As we discussed in earlier chapters, the CSK dynamically loads a skin (ASCX) file that lays out the controls for a particular theme. If you need to programmatically interact with any of the controls on a skin, you'll need to obtain a reference to the control. When editing an FAQ we will need to obtain the contents of the `TextBox` object holding the FAQ question. You can obtain references to controls using the `GetControl` and `GetOptionalControl` methods implemented in the `SkinnedComunnityControl` base class. There are additional controls we will need reference to, but only the question `TextBox` is shown here:

```
protected override void OnLoad(EventArgs e)
{
  if (!Page.IsPostBack)
  {
    ContentPageID = Int32.Parse(
      Context.Request.QueryString["id"]);

    FaqInfo faqInfo =
      (FaqInfo)FaqUtility.GetFaqInfo(
      objUserInfo.Username, ContentPageID);

    EnsureChildControls();
    txtAnswer.Text = faqInfo.AnswerText;
    dropTopics.SelectedTopicID = faqInfo.TopicID;
    txtIntro.Text = faqInfo.IntroText;
    txtQuestion.Text = faqInfo.QuestionText;
    txtReference.Text = faqInfo.ReferenceText;
  }
}
```

When the page loads, we need to retrieve the information for an existing FAQ from the database. The CSK will pass the content identifier in the query string parameters, so we fetch the ID and pass it along to the `GetFaqInfo` method of the `FaqUtility` class we examined earlier. Once we have an `FaqInfo` object in hand, we can populate controls on the page (which we obtained references to during the skin load event) with FAQ information.

```
void PreviewFaq(Object s, EventArgs e)
{
  if(objSectionInfo.EnableTopics)
    topicPreview.Name = dropTopics.SelectedItem.Text;
  questionPreview.Text = txtQuestion.Text;
  introductionPreview.Text = txtIntro.Text;
  answerPreview.Text = txtAnswer.Text;
  referencePreview.Text = txtReference.Text;
}
```

When the user clicks the preview button, we need to shuttle all of the content in the edit controls to the preview controls, which will render the content with the styles we use to display content. This gives the author a better idea of how the content will look when the content goes live. The `ContentEdit` base class will take care of toggling the preview panel control visibility so the author can see the results.

```
void SubmitFaq(Object s, EventArgs e)
{
  if (Page.IsValid)
  {
    // Get Topic
    int topicID = -1;
    if (objSectionInfo.EnableTopics)
      topicID = Int32.Parse(dropTopics.SelectedItem.Value);

    FaqUtility.EditFaq(
      objUserInfo.Username,
      objSectionInfo.ID,
      ContentPageID,
      topicID,
      txtQuestion.Text,
      txtIntro.Text,
      txtAnswer.Text,
      txtReference.Text);

    Context.Response.Redirect(CommunityGlobals.CalculatePath(
      String.Format("{0}.aspx", ContentPageID)));
  }
}
```

The `SumbitFaq` event handler uses the `FaqUtility` class to put the updated content into the database. Once this is done, we send the user off to the content page with a `Response.Redirect` to view the updated FAQ.

```
int ContentPageID
{
  get { return (int)ViewState["ContentPageID"]; }
  set { ViewState["ContentPageID"] = value; }
}
TextBox txtQuestion;
TopicPicker dropTopics;
TextBox txtIntro;
HtmlTextBox txtAnswer;
HtmlTextBox txtReference;
DisplayTopic topicPreview;
Title questionPreview;
BriefDescription introductionPreview;
FaqAnswer answerPreview;
FaqReference referencePreview;
string _skinFileName = "Faqs_AddFaq.ascx";
string _sectionContent =
  "ASPNET.StarterKit.Communities.Faqs.FaqSection";
}
```

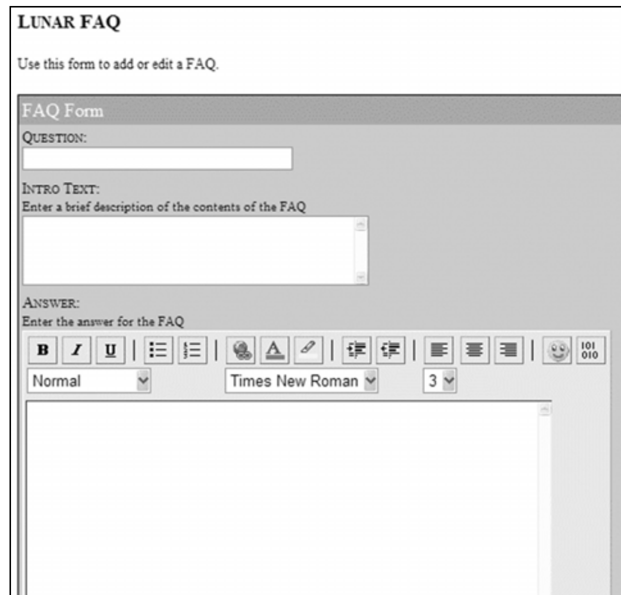
The `AddFaq` page looks similar to the `EditFaq` page and can be seen in the code download. The only work left is in presenting the data via skins and style sheets.

FAQ Page Content Skins

Our FAQ module requires three skins:

- A skin to display a single FAQ in detail
- A skin to display a list of FAQs for a section
- A skin to add or edit an FAQ

At a minimum, we need to create these three skin files in the default theme for the communities. We can add additional variations of the skins to the other themes if we want the FAQs to take on a different appearance. For adding and editing an FAQ, we would like the screen to look like the following:



Make sure the filename for the skin matches the filename assigned to `SkinFileName` in the content page class. You'll also need to pay special attention to control names, as these must match the control names you search for with `GetControl` in the underlying class.

The easiest way to get started is with an existing skin from a working module, because you'll also need to match up the controls to the base class. Remember that our `AddFaq` class derives from the `ContentAddPage`, which expects certain controls on the form, such as a panel named `pnlForm`, and a button named `btnAdd`. Let's look at an excerpt from the `Faq_AddFaq` skin:

```
<%@ Control %>
<%@ Register TagPrefix="community"
```

```
        Namespace="ASPNET.StarterKit.Communities"
        Assembly="ASPNET.StarterKit.Communities" %>
    <community:SectionTitle
        CssClass="Form_Title"
        Runat="Server"
        ID="Sectiontitle1"
        NAME="Sectiontitle1"/>
    <p class="Form_Description">
      Use this form to add or edit an FAQ.
    </p>
    <asp:Panel id="pnlForm" Runat="Server">
    <TABLE cellSpacing="0" cellPadding="3" width="520" class="Form_Table">
      <TR>
        <TD class="Form_SectionRow">
          FAQ Form
        </TD>
      </TR>
      <tr class="Form_LabelRow">
        <td >
          <span class="Form_LabelText">Question:</span>
          <asp:RequiredFieldValidator
               ControlToValidate="txtQuestion"
               Text="(Required)"
               Runat="Server"
               ID="Requiredfieldvalidator1"
               NAME="Requiredfieldvalidator1"/><br>
          <asp:TextBox id="txtQuestion"
               CssClass="Form_Field"
               columns="40"
               runat="server">
          </asp:TextBox>
        </td>
      </tr>
      ...
```

Note the use of a `RequiredFieldValidator` to ensure the FAQ will have a question populated. The `ContentAddPage` class will also expect a preview panel, which comes later in the `Faq_AddFaq.ascx` file.

```
    ...

    <asp:Panel id="pnlPreview" Runat="Server">
    <table cellpadding="5" width="520">
    <tr>
        <td align="right">
        <community:DisplayTopic id="topicPreview" runat="Server"/>
        </td>
    </tr>
    <tr>
        <td>
        <community:Title id="questionPreview" Runat="Server"/>
        </td>
    </tr>
    <tr>
        <td>
        <community:BriefDescription id="introductionPreview"
                        Runat="Server" />
        </td>
    </tr>
    <tr>
        <td>
        <community:FaqAnswer id="answerPreview" Runat="Server" />
        </td>
    </tr>
```

```
<tr>
    <td>
    <community:FaqReference id="referencePreview" Runat="Server" />
    </td>
</tr>
</table>
<p>
<asp:Button id="btnContinue" Text="Finish Preview" Runat="Server" />
</asp:Panel>
...
```

The skins to display content are easier to build since you just need to lay out the display controls as you see fit. The base content display pages will match up the controls for you. Here is the skin to display a single FAQ:

```
<%@ Control %>
<%@ Register
    TagPrefix="community"
    Namespace="ASPNET.StarterKit.Communities"
    Assembly="ASPNET.StarterKit.Communities" %>
<table width="100%" cellspacing="0" cellpadding="11" class="Faq_Table">
    <tr>
        <td class="Faq_IntroCell">
            <div align="right">
                <community:DisplayTopic Runat="Server"
                    ID="Displaytopic1" NAME="Displaytopic1" />
            </div>
            <community:Title Runat="Server" ID="Title1"
                           NAME="Title1" />
            <br>
            <br>
            Posted by
            <community:Author CssClass="Faq_AuthorLink"
                Runat="Server" ID="Author1" NAME="Author1" />
            on
            <community:DateCreated Runat="Server"
                ID="Datecreated1" NAME="Datecreated1" />
            <br>
            <br>
            <community:BriefDescription Runat="Server"
                ID="Introduction1" NAME="Introduction1" />
        </td>
    </tr>
    <tr>
        <td class="Faq_AnswerCell">
            <community:FaqAnswer Runat="Server"
                ID="Answer1" NAME="Answer1" />
        </td>
    </tr>
    <tr>
        <td class="Faq_AnswerCell">
            <community:FaqReference Runat="server"
                ID="Reference1" Name="Reference1" />
        </td>
    </tr>
    <tr>
        <td class="Faq_BodyCell">
            <br>
            <community:Rating SubmitText="Rate Item"
                Runat="Server" ID="Rating1" NAME="Rating1" />
        </td>
    </tr>
</table>
```

```
<table width="100%" cellspacing="0" cellpadding="11">
    <tr>
        <td>
            <div class="Content">
                <community:Notify
                    Text="Notify me when a new comment is posted"
                    Runat="Server" ID="Notify1" NAME="Notify1" />
                <p>
                    <community:Comments Runat="Server"
                        ID="Comments1" NAME="Comments1" />
                <p>
                    <community:FaqEditContent
                        CommentText="Add Your Comment"
                        EditText="Edit this FAQ"
                        DeleteText="Delete this FAQ"
                        Runat="Server" ID="Faqeditcontent1"
                        NAME="Faqeditcontent1" />
            </div>
        </td>
    </tr>
</table>
```

In this skin, we use all of the web controls we built (including `FaqEditContent` which only displays links appropriate to the type of user viewing the FAQ). With the skins in place, we are only one step away from completing our new module.

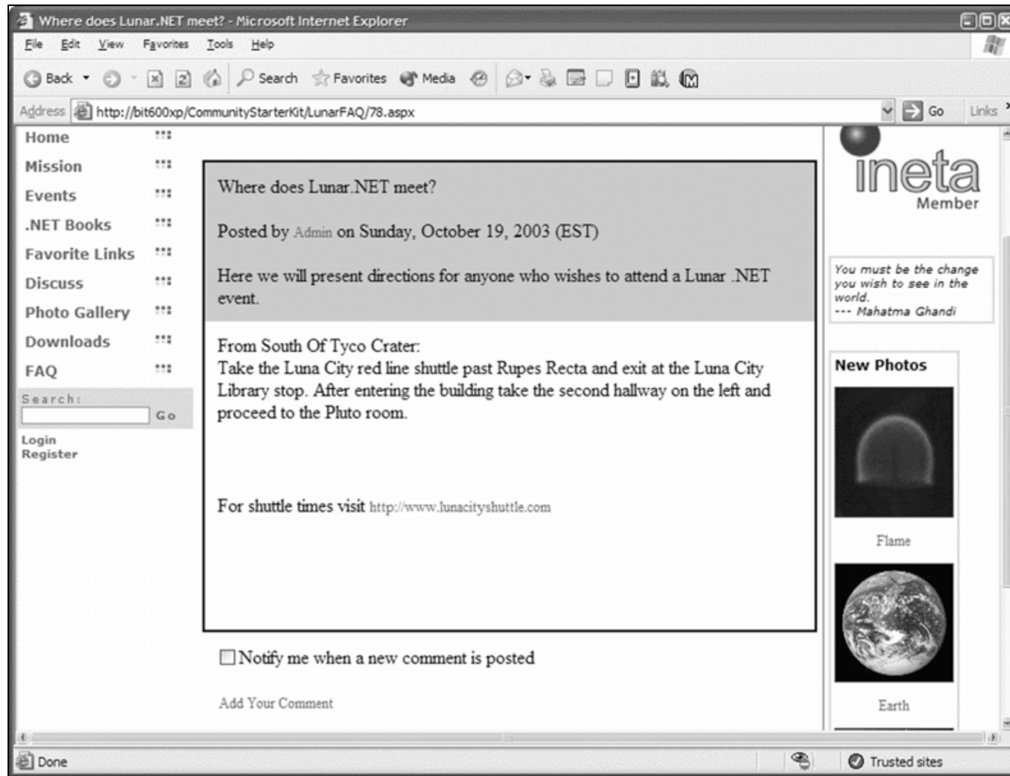
Module Styles

While building the skins and web controls for a module, keep track of the different CSS rule names you place into the code. The time has finally come to modify CSS files to put the new styles required by the module into place. Unfortunately, the CSK doesn't provide any default behavior for stylesheet selection, so you will need to place these styles into every stylesheet available to a community.

> Copy styles from an existing module and just change the names.

Putting It Together

At this time, you can start testing the new module. The working result of the FAQ module is displayed in the following image:



Summary

Building a new module requires building on top of existing code in the CSK, so a good understanding of the CSK architecture is in order. By following the steps we outlined in the beginning of this chapter and following the patterns set forth by the existing modules, you can build a new module with the ability to search and rate the module content. It will also have all of the other cross-cutting functionality built into the CSK.

In the next chapter, we will build a little more onto the FAQ module and take a look at other forms of customization in the CSK. Although building a new module requires a bit of code and time, the end result is a seamlessly integrated feature with the full support of the CSK comments, ratings, view counts, and more.