CCS Developer

AUGUST 2004

VOLUME NO. 1 ISSUE NO. 1

DataObjx L.L.C.

Website: www.dataobjx.net

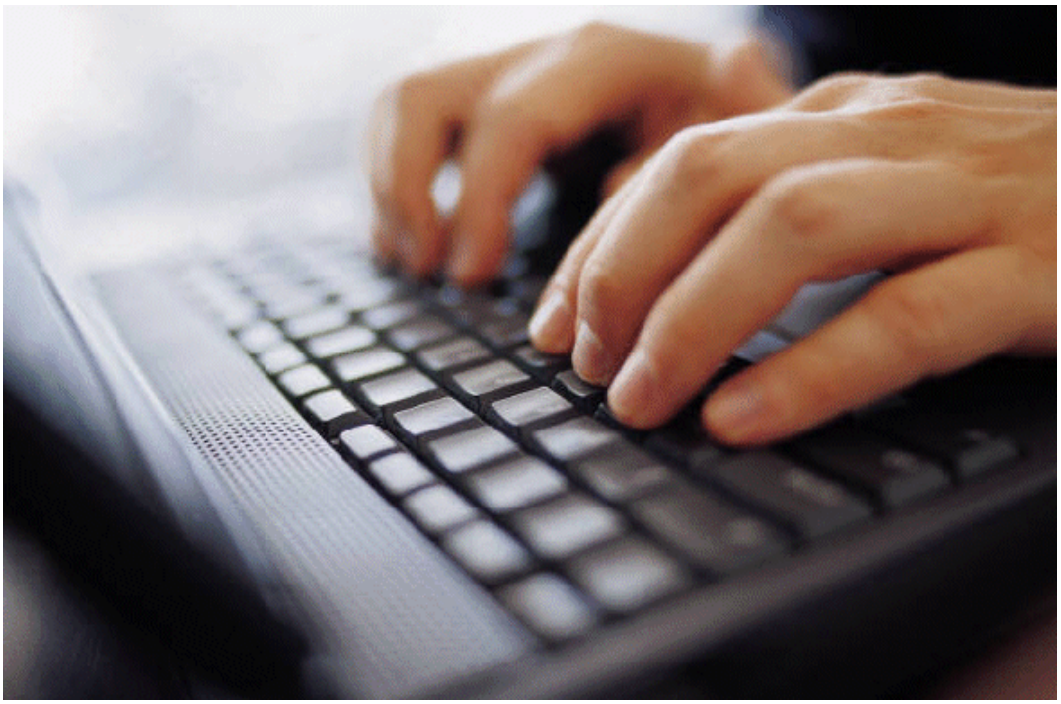
DataObjx L.L.C.
Meriden, CT USA

Office: 860/919-2536

Email: administrator@dataobjx.net

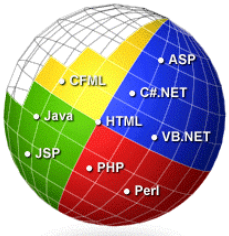
Website: www.dataobjx.net

DataObjx L.L.C.



Do you have an article to contribute?

Contact DataObjx to find out how.



Need More Answers?
Click here to access the Code Charge Studio Forums

In This Issue

Editors Comments

Viewpoint...

Welcome to the first issue of the **CCS Developer Magazine**.

We've been busy trying to create the first magazine dedicated to Code Charge Studio Developers and we hope you enjoy our first issue.

In this issue, we begin with a number of articles geared toward making you a better CCS developer and hopefully, making your products better as well.

Fortunately for us, CCS can generate code in many languages, however our first edition primarily covers .ASP code. We apologize for this, but like anything new, one has to start somewhere.

You will notice that there are requests everywhere in this issue asking for your input and hopefully, you will respond with applications and techniques that you've created or discovered.

If you program in PHP, .NET, PERL or any of the other languages that CCS can generate, and you're interested in supporting the CCS community, send us your information, code and project files so we can publish them along with our articles.

This introductory issue is free, but in order to continue publishing **CCS Developer Magazine**, we need your support and your subscription.

Our goal while creating this informative magazine was "if this issue saves you an hour or more, we'll have done our job well". In this issue, we show you how to integrate WYSIWYG editors into you applications, build a re-useable file upload wizard that allows you to upload files directly into your database as well as a number of useful techniques to aid you along the way to building better web applications with CodeCharge Studio.

The **CCS Jobs Board** is open for business, so if you have a project you want other CCS Developers to bid on and fulfill, this is the place to advertise it. A more advanced CCS Job Board is nearly complete and will go live soon enhancing your project seeking experience.

In addition, we'll be launching the **CCS Project Marketplace** next week that will allow you to sell your CCS Project Source Code and/or Applications on-line.

We will also be providing an article on integrating the ASP Report Wizard by ASPWebSolution with your CodeCharge Studio Projects.

We'll continue to establish relationships with third-party vendors that offer components or applications that can be integrated with CodeCharge Studio and we're interested in knowing what your interests are. To that end we've started to offer you more user preferences. These will be expanded to include the languages you program in and so on in order help you have a more enjoyable programming experience.

Your user preferences will also allow the CCS Jobs Board to interact with you by matching project programming languages, database and other requirements with your capabilities so you'll be notified when a project meeting your capabilities has been advertised.

We have space allocated to advertise commercial products you have created with CCS, so contact us to utilize this space if you have a CCS products you want people to know about.

We're also interested in hearing from you regarding the types of articles you'd like to see in up-coming issues of CCS Developer Magazine. Don't hesitate to email us at administrator@dataobjx.net.

To subscribe, simply login and click the subscribe link contained on the CCS Developer Magazine landing page.

We hope that you find the articles within this issue of interest. Moreover, we hope that some of the techniques you learn from this issue help you in developing your application for the CodeCharge Studio Developer Award. Submissions are underway now. Results for the competition will be announced on October 1, 2004.

We hope you enjoy reading CCS Developer Magazine as much as we did creating it – see you on-line.

Best Regards,

Martin Hamilton
www.dataobjx.net

CodeCharge Studio

the fastest way to develop web applications

FEATURES

Visual IDE and Code Generator

Rapidly build web applications by generating robust, scalable programming code in ASP.NET (C# & VB), ASP, PHP, Java, ColdFusion and Perl.

Database-to-Web Converter

Convert any database into a Web application in just a few minutes.

MS FrontPage® Add-In

Transform Microsoft® FrontPage® into a powerful Web development environment.

Pre-built Solutions

Customize and enhance any of the examples included with CodeCharge:

- Community Portal
- Online Store
- Task Management System
- Employee Directory
- Bug Tracker
- Forum / Message Board

BENEFITS

Reduce Development Time

Eliminate time-consuming programming tasks and build scalable, robust Web Applications in a fraction of the time. and Perl.

Minimize Errors

Avoid costly programming errors and misspellings by generating consistent, well-structured code.

Reduce Learning Curve

Analyze and modify generated code to learn web technologies and take on programming projects in any environment.

www.codecharge.com

"Unfortunately, it often doesn't matter whether you are a small to medium sized business - or for that matter a large organization… every organization needs or wants a centralized point where-in documents can be stored and the 'versions' of those documents 'controlled'."

Formatting Row Colors Dynamically.

Changing row color according to database values.

Language: .ASP

There are many reasons why we would like various records to be displayed with a different background color based on a data value.

Why would you want to do this?

One every-day example is in Outlook type applications where we want the article or message title to appear one way if not read and perhaps in a different color if it has been read.

This makes it easier to determine which articles/messages you have *yet* to read.

In another example, lets say you have a record set comprised of 4 records.

The record-set has a field called "risk".

Record 1 has a "risk" = 1
Record 2 has a "risk" = 2
Record 3 has a "risk" = 3
Record 4 has a "risk" = NULL

When a row is written to the grid, we want any records with a

- a "risk" of 1 to be in Red (High risk)
- "risk" of 2 to be in Orange. (Medium risk)
- "risk" of 3 to be in Yellow. (Low Risk) and
- any records with a "risk" of NULL to be in whatever the standard theme is... e.g., (No Risk).

Continued on page 3

FRCD – Figure 1

```
<style>
.RedDataTD { background-color: #FF0000; color: #FFFFFF; font-size: 13px;
             border-left: 1px solid #C6C7BD; border-right: 1px solid #EFEBE7;
             border-top: 1px solid #C6C7BD; border-bottom: 1px solid #EFEBE7 }
.OrangeDataTD { background-color: #FF9900; color: #000000; font-size: 13px;
             border-left: 1px solid #C6C7BD; border-right: 1px solid #EFEBE7;
             border-top: 1px solid #C6C7BD; border-bottom: 1px solid #EFEBE7 }
.YellowDataTD { background-color: #FFFF00; color: #000000; font-size: 13px;
             border-left: 1px solid #C6C7BD; border-right: 1px solid #EFEBE7;
             border-top: 1px solid #C6C7BD; border-bottom: 1px solid #EFEBE7 }
</style>
```

Dynamic Style Sheets

Incorporating Dynamic CCS Style Sheets

Ever wondered how to change the CCS Themes for your site 'on the fly'?

In this article, we'll show you how it's done with a minimum of code.

Locate the Get Content() Function

Open your CCS project and then double click on "Common Files".

Then, click on the "Cache.asp" tab.

Next, locate the block of code shown in DSS – Figure 1.

DSS - Figure 1

```
Public Default Property Get Content()

  If NOT IsOpen Then
    mContent = GetFileContent(mFSO, mName)
    IsOpen = True
  End If

    Content = mContent

  End Property
```

We've chosen the 'Get Content()' routine as the most appropriate place to put our code.

Continued on page 5

"Yes Software anticipated the developers desire to create their own components and therefore CCS has the ability to detect your component directory and your custom components.
Your components are then displayed within the CCS Toolbar, just like regular CCS components"

You can write for CCS Developer Magazine!

If you have
- tips or tricks,
- developed a CCS application,
- integrated a third-party product with CCS

We'd like to hear from you.

Contact DataObjx now to have your article published in CCS Developer Magazine.

Share Your Knowledge

We're looking for developers using Code Charge Studio to contribute articles of interest to the readers of **CCS Developer Magazine**.

If you can contribute an article, tip, trick or even a routine or two – we are interested in hearing from you.

Click here to send an email to DataObjx and let us know what you can contribute to **CCS Developer Magazine**.

CCS Components: The CCDLookup Component

Building A CCS Component

Language: Misc

In this article, we'll see how we can create a CCS Component that will quickly write out the CCDLookup function, a commonly used function.

This same technique can be used to create your own CCS Components to further increase your speed of development.

Yes Software anticipated the developers desire to create their own components and therefore CCS has the ability to detect your component directory and your custom components. Your components are then displayed within the CCS toolbar, just like regular CCS components.

Step 1 – Create The 'Custom' Directories

We're going to add a sub-directory to hold the files we are going to create. The name that we use to create that directory is the name that will appear as a tab on the toolbar. Therefore, we're going to create a relatively 'short' name that has a minimum of characters.

Locate the default Code Charge Studio Directory. The default directory is;
c:\program files\codechargestudio\

Next, locate the \Components sub-directory.

[NOTE: If you changed the default directory when you installed Code Charge Studio, locate that directory accordingly].

Under the \Components directory, locate the "\ToolBox\" directory. This is where the Code Charge Studio component files are held.
[c:\program files\codechargestudio\Components\Toolbox\]

Create a sub-directory on the \ToolBar\ branch called "Custom".

Naturally, feel free to name the directory as you see fit. But keep it short and if you do change it, use your directory name in place of the one used in this article ("Custom").

Now, select the "\Custom" directory and create two sub-directories under it. Name one of the sub-directories "\icon" and the other sub-directory "\js".

The "\icon" sub-directory will hold the .ico (icon) files that will be displayed on the "\Custom" tab within the toolbox.

The "\js" sub-directory will hold the JavaScript files that CCS will use to produce the code that the component represents.

Step 2 – Create The Custom.xml File

When Code Charge Studio initializes, it reads the .XML files in the "Toolbox" directory and uses the information in the .XML files to locate the component_name.xml files.

We need to create the .XML file for the "Custom" directory that we just created to tell CCS what directory to look into - to load the component files that we're about to create.

Open Notepad and add the code shown in CSSC-Figure 1.

CSSC – Figure 1	Custom.xml
<item name="Custom" number="5" hint="Development Extensions" folder="Custom" icon="Custom.ico"/>	

Save this file as custom.xml in the "Toolbox" directory.

Before moving on, we have two (2) parts of the 'code' to look at.

Continued on page 4

Formatting Row Colors Dynamically

Changing row color according to database values, continued….

…Continued from page 1

We've created some sample styles to accommodate our example. (Refer to FRCD – Figure 1)

You will need to paste these styles before the </head> section within the HTML page.

This article covers one of the techniques used to achieve this using Code Charge Studio.

We will use the "Products" table that comes with the Northwind Database.

Create A New Project

Open a new project. For our example application, we have called the application; "RowColors".

We're going to choose a simple routine to encapsulate within our component so that we can focus on the mechanics of getting your component up and running.

In our next issue, we're going to build a Component Maker to really simplify the process.

Set-up The Database Connection

Set up a connection to the Northwind.mdb database that installs by default when you install Microsoft Access.

Create The Default Page

Now, add a new page called "default".

Add The Styles

Add the styles to the HTML page that contains the grid (default.html).

FRCD – Figure 2

List of Products

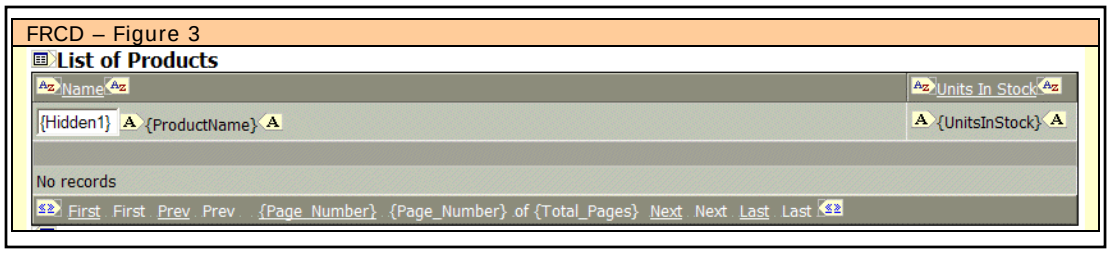
Name	Units In Stock
Alice Mutton	0
Aniseed Syrup	13
Boston Crab Meat	123
Camembert Pierrot	19
Carnarvon Tigers	42
Chai	39
Chang	17
Chartreuse verte	69
Chef Anton's Cajun Seasoning	53
Chef Anton's Gumbo Mix	0

First Prev 1 2 3 4 5 6 7 8 of 8 Next Last

Add The Products Grid

Next, add a Grid to display the Products table. When prompted to select a table, select the Products table. We will display two (2) fields from the Products table, namely the Product Name field and the number of Units In Stock field.

FRCD – Figure 3



What we want is to display any products that have

- 5 or less units in stock to be displayed with a red background;
- products that have 20 or less units in stock to be displayed with an orange background;
- products that have 21-50 units in stock to be displayed with a yellow background, and
- all products with 51 or more units in stock to be displayed with the normal CCS Style.

In our example we only use one row rather than alternating rows.

Continued on page 7

Tip
When you attempt to write a component that is going to write out .asp, .net, .php, etc. there are a few rules that you have to abide by or you will get an error when you attempt to paste the code into the HTML of your page.

Tip
In JavaScript "Placing a **..** (dot dot backslash) in front of a directory path causes the internet server to traverse 'Up' one directory from the current directory… for each (dot dot backslash) Typed .

Also, \n (backslash followed by the letter "n") tells the pasteHTML function to perform a carriage return."
A Quote has to be replaced with a \" (backslash – quote)

CCS Components: The CCDLookup Component

Building A CCS Component, continued…

First, notice the part that says [number="5"]. This is telling Code Charge Studio that this will represent the 5th tab. You may need to change this number to a "4" or perhaps a "6", depending on how many other component tabs you currently have.

How do you know what number to use?

It's pretty logical once you look at it. Open the Builders.XML file and look at the number it has… a one (1). If you look at your Toolbox within Code Charge Studio, you'll notice that the "Builders" tab is indeed the first tab in the Toolbox.

Now, open the Forms.XML file and you'll notice that its number is two (2). Again, it's the second tab in the Toolbox.

We could go on opening and reading each file – [HTML.XML is number three (3)], but we don't need to now that we understand how CCS is using "number="#" within these files.

Open Code Charge Studio and count the number of tabs you have in your Toolbox. For instance, if there are only three tabs, you'll want to set the number in your "Custom.XML" file to four (4).

The second thing we need to notice in the file is the [icon="Custom.ico"] part. This is telling CCS what icon to display in the Toolbox tab. Locate an icon of your choice to represent your "Custom" tab. If you can't find one right away, you can use the one supplied in the source code for this article.

To sum up, you've created a .XML file and you have saved it to the Toolbox directory. In addition, you've copied an icon to the Toolbox directory and you've modified the code in the .XML file accordingly.

To see your "Custom" tab appear, exit Code Charge Studio if you already have it open and then re-start Code Charge Studio.

If everything is correct, you should see your "Custom" tab appearing in the Toolbox (within CCS).

Of course, we haven't added any components to our "\Custom" directory, so if you click on the tab – it's empty.

Let's create our first component….

Tips & Tricks
"When you try to use a component in an events (code) page, CCS will not let you, instead providing a warning that says, "Toolbox is only available in the design or HTML mode."
But you can still write native language routines (.asp, .net, .php, etc…) paste them into the bottom of your HTML page, then cut and paste them into the events (code) page.
It can save a lot of typing."

Creating The CCDLookUp.XML File

Now we're ready to create our CCDLookUp.XML file. This file will contain the code

Open Notepad and insert the following code:

CSSC – Figure 2	CCDLookUp.XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<item name="CCSLookUp" number="1" hint="Perform a CCS LookUp"
script="..\..\..\Components\ToolBox\custom\js\CCDLookUp.js"
img="..\..\..\Components\ToolBox\custom\icon\CCDLookUp.ico"/>
```

Remove any line breaks on the second (2nd) line (**<item name… />**) that may result from a copy/paste, it should be one line.

Save this file to the "\Custom" sub-directory as "CCDLookUp.XML". Save an icon called "CCDLookUp.ico" to the \Custom\icon directory. If you don't have an icon on hand you can

Continued on page 9

Dynamic Style Sheets

Incorporating Dynamic CCS Style Sheets, continued…

This routine is responsible for getting the content that will be streamed to the users browser.

We're going to intercept that code and replace the styles dynamically.

Add the code between the "Dynamic Theme Code Begins …. Ends as shown in DSS – Figure 2.

DSS – Figure 2

```
Public Default Property Get Content()

   If NOT IsOpen Then

     mContent = GetFileContent(mFSO, mName)

'DYNAMIC THEME CODE BEGINS
'DynamicTheme() Function located in Common.asp

          mContent = DynamicTheme(mContent)

'DYNAMIC THEME CODE ENDS

     IsOpen = True

   End If

   Content = mContent

 End Property
```

Create The Dynamic Theme Function

Now, we're going to create the DynamicTheme() function and we're going to add it to the bottom of Common.asp.

From the tab, click on "Common.asp" then scroll all the way to the bottom of the page and add the following block of code….

DSS – Figure 3

```
CONST DefalutCSSTheme  = "Olive"
CONST DefaultCSSPath     = """/Themes/Olive/Style.css"""
'When The User Logs In, Get the ThemeName and CSSPath from the User preference
'or company table if you have one.

Function DynamicTheme(vContent)
Dim sThemeName
Dim sNewThemeName
Dim sContent

        If Len(vContent) = 0  Then Exit Function

        sContent                = vContent
        sThemeName              = "class=""" & DefalutCSSTheme
        sNewThemeName           = "class=""" & Session("CompanyThemeName")

        'First, Replace the path to the default style sheet.
        sContent = Replace(sContent, DefaultCSSPath, Session("CompanyCSSPath"))

        'Next, Replace all references to the name of the old styles to the
        'names of the new styles
        sContent = Replace(sContent, sThemeName, sNewThemeName)

        DynamicTheme    = sContent
End Function
```

Continued on page 6

"Notwithstanding our need to have adequate programming skills to create the application, we also need to be concerned about how the user will interact with the system."

Dynamic Style Sheets

Incorporating Dynamic CCS Style Sheets, continued…

About The Dynamic Theme Function

When the page content is loaded into the Get Content() [cache.asp] property executes it sends that content to our DynamicTheme function.

It then performs a 'Search and Replace' on our theme path and theme name within 'vContent' and when finished it returns the modified code back to the Get Content method.

[mContent = DynamicTheme(mContent)]

Defining The Constants

There are two constants defined.

One for the theme name and one for the stylesheet name.

When we develop a CCS application, we tend to use one style. However, that may not be the style the user prefers.

Still, we have to tell the application what style name to replace and what style sheet (.css) to replace when the function is called.

Since this rarely changes for your entire project, the DefaultCCSTheme and the DefaultCCSPath have been defined as constants.

Why?

Because when you publish your site up to the internet, you have a base theme that you used when you built the pages.

Thus, we'll define these as Constants. However, if you create another project an use a different theme and style sheet, don't forget to replace the constant values appropriately.

Getting The Users Preferred Theme

Now that we have the majority of code in place, we need to amend the login code.

For the purpose of this example, we're going to assume that we are going to allow the user to select and save to the users table, their preferred theme.

Therefore, we're going to add a field to the Users table called "Preferred_Theme" and "Preferred_Style_Sheet" then we're going to modify the Function CCLoginUser(Login, Password) accordingly.

First, add the two fields to your users table.

Next, locate the CCLoginUser function in Common.asp and add the field to your select statement…

SQL = "SELECT users.user_id Preferred_Theme, Preferred_Style_Sheet….

Next, we need to create the session variable used by the DynamicTheme() function.

```
Session("CompanyThemeName")    = "DeepWater"
Session("CompanyCSSPath")=
"http://www.dataobjx.net/Themes/DeepWater/Style.css"
```

Add the following lines of code to the Session variables section of the code.

Remove any line breaks so that both lines are on one line each.

How it works…

Essentially, the routines work like this;

When the user logs in, their 'theme preference' is determined and the session variable for the theme-name and theme-path are written to the users session.

Then, as the page begins to load, the Get Content property is executed and the HTML output is rendered into the "mContent" variable.

However, before the routine is finished – we've intercepted the code and performed a search and replace on the necessary tags.

Our modified output is then sent to the users browser and the styles are presented according to the preferences they saved.

We are creating two fields because there may be times where the Theme Name can conflict with content on the page.

In this example, we're using the "Olive" theme. If the website contained information about a restaurants menu, the word "Olive" could be replaced by mistake if we didn't do this. In other words, you couldn't simply perform a search and replace on the word olive… we need to perform a search and replace on the phrase "class="Olive…" instead.

Also, we've used a full http address on the style sheet because if you decide that you will host an application, the company purchasing time on the application may want their own style sheet to be used.

Dynamic Style Sheets

Incorporating Dynamic CCS Style Sheets, continued…

FRCD – Figure 4

```
Function Products_Hidden1_BeforeShow() 'Products_Hidden1_BeforeShow @8-790308A7

'Custom Code @10-73254650
' -----------------------
' Write your own code here.
Dim NumUnitsInStock
Dim sStyle

        sStyle = "OliveDataTD"     'Default Style

        NumUnitsInStock = Products.datasource.recordset.fields("UnitsInStock")

        'handle any null values
        If Len(NumUnitsInStock) = 0 Then NumUnitsInStock = 0

        If NumUnitsInStock < 6 Then
                sStyle = "RedDataTD"
        ElseIf NumUnitsInStock > 5 And NumUnitsInStock < 21 Then
                sStyle = "OrangeDataTD"
        ElseIf NumUnitsInStock > 20 And NumUnitsInStock < 51 Then
                        sStyle = "YellowDataTD"
        End If

        Products.Hidden1.value = sStyle
' -----------------------
'End Custom Code

End Function 'Close Products_Hidden1_BeforeShow @8-54C34B28
```


Add A Hidden Field

Next, we're going to add a hidden field in front of the Products Label, (See FRCD – Figure 3).

Select the Hidden1 field component and from the Properties select the 'Events' tab.

Add Code To The Before Show Event

Then right click on 'Before Show' and add click 'Add Code'.

We're going to place some code into the before show event that will cause a style name to be written into the Hidden1 field for each record retrieved.

The code is displayed in FRCD - Figure 4.

Step 3

Now we simply need to modify the HTML page by replacing the class="OliveDataTD" to class="{hidden1}".

The original code will appear similar to that shown in FRCD – Figure 5

FRCD – Figure 5

```
<!-- BEGIN Row -->

<tr>

        <input style="WIDTH: 65px; HEIGHT: 22px" type="hidden" size="8" value="{Hidden1}" name="{Hidden1_Name}">

  <td class="OliveDataTD">{ProductName} </td>

  <td class=" OliveDataTD " width="1%">{UnitsInStock} </td>

</tr>

<!-- END Row -->
```

Continued on page 8

Dynamic Style Sheets

Incorporating Dynamic CCS Style Sheets, continued…

Now, replace the class="OlveDataTD" with class="{hidden1}" as shown in FRCD – Figure 6;

FRCD – Figure 6

```
<!-- BEGIN Row -->
<tr>
        <input style="WIDTH: 65px; HEIGHT: 22px" type="hidden" size="8" value="{Hidden1}" name="{Hidden1_Name}">
  <td class="{Hidden1}">{ProductName} </td>
  <td class="{Hidden1}" width="1%">{UnitsInStock} </td>
</tr>
<!-- END Row -->
```

Now with that done, re-generate the page and save your work; then open the page in your browser.

The result should appear similar to that shown in the FRCD – Figure 7.

FRCD – Figure 7

List of Products

Name	Units In Stock
Singaporean Hokkien Fried Mee	26
Sir Rodney's Marmalade	40
Sir Rodney's Scones	3
Sirop d'érable	113
Spegesild	95
Steeleye Stout	20
Tarte au sucre	17
Teatime Chocolate Biscuits	25
Thüringer Rostbratwurst	0
Tofu	35

First Prev 1 2 3 4 5 6 7 8 of 8 Next Last

Admittedly the colors we've used in the example are a bit extreme, but the example is clear.

Using this technique you can allow your users to evaluate the status of a product rapidly, simply by looking at the color of the row.

Now that you know how easy it is to provide this functionality, you may see occasion to use it in one of your own projects.

CCS Components: The CCDLookup Component

Building A CCS Component, continued…

Continued from page 4….

use the icon that comes with the source code for this article.

Before we continue, let's have a look at the code within this .XML file.

There are 5 parts of the code we should look at and understand. Refer to CCSC – Figure 3.

CCSC – Figure 3	
item name="CCSLookUp"	Tells CCS the name of the componet
number="1"	Tells CCS the sequence number or sort order that the component will appear within your "Custom" tab.
hint="Perform a CCS LookUp"	Tells CCS what 'tooltip' should appear when you hover your mouse over the component.
script="..\..\..\Components\ToolBox\custom\js\CCDLookUp.js"	Tells CCS where the code is that will be 'generated' by CCS when you work with that component.
img="..\..\..\Components\ToolBox\custom\icon\js.ico"	Tells CCS what icon to display to represent this component.

The 'script' and 'image' parts of the code have a number of "..\ - dot dot backslashes" at the front of the line. Essentially each – dot dot backslash (..\) tells the internet server to traverse up one directory for each – dot dot backslash.

Don't worry about those for the moment however, just leave them as they are and they will work as anticipated.

Notice that the 'script' line tells CCS where to locate the CCDLookUp.js file – this is the file that contains the code that CCS will paste into our application when we use the component.

So far, we've told CCS to create a tab called "Custom" in the Toolbox. Then we've told CCS to display an icon to represent a component called CCDLookUp and to make it the first component within the tab.

We've also told it what icon to use.

Create The Component

Now we're ready to create the CCDLookUp.js file. This is the file that contains the code that will be written to your project file when the component is selected.

Open a new instance of Notepad, paste the following code into it and save the file to the "\Custom\js" directory as CCDLookUp.js.

CCSC – Figure 4	CCDLookUp.js

```
#include ..\..\..\Dialogs\Common\Common.js

var ccObject = ccPage.Project;

pasteHTML("<!--  \n '********** ==> Routine Begins <== **********\n'This routine is used to derive a value from a table in the database. \nDim sResult \nDim lngID\n  lngID = Request.QueryString(\"id\")\n  If  Len(lngID) > 0 Then \n    sResult = CCDLookup(\"FIELD_NAME\", \"TABLE_NAME\", \"id=\" & CCToSQL(lngID, \"Integer\"), DBIntranet)\n  End If\nFORMNAME.LABEL.value = sResult\n'********** ==> Routine Ends <== ********** \n//--> \n");
```

The above code is comprised of three (3) lines. The third (3rd) line is wrapping into many lines, but it is actually only one line of code, therefore make sure that the line beginning with pasteHTML is all on one line.

Now we'll save the files, but don't close them down… we're going to refer to them in a moment.

Save this file to the "\js" sub-directory as "CCDLookUp.js".

So you should now have one file called CCDLookUp.XLS in the "\Custom" directory and one file called "CCDLookUp.js" in the "\Custom\js" sub-directory.

And you should have an icon in the \Custom\icon directory called CCDLookUp.ico.

If you have found a different icon or want to use a different name for the icon make sure you edit the following line in the CCDLookUp.XLS file…

img="..\..\..\Components\ToolBox\custom\icon\your_icon_file_name.ico"

replacing the name of the your_icon_file_name.ico file accordingly.

Continued on page 11

Building A Document Management System

FEATURED ARTICLE: Part 1 of a three part series…

Language: .ASP
Data Base: MS Access

In this series, we're going to build a document management system.

Initially, we'll build the framework for the application. Then we'll add additional functionality like allowing users to check-in and check-out documents, and finally, we'll add finishing touches to the application.

During this series of articles, we'll include many of the features and functionality that we discuss in other articles in future issues of CCS Developer Magazine.

Why Build A Document Management System?

As developers we're generally used to such systems to handle revision control of source code.

But if we look around us, the current business that we work for struggles with 'change control procedures'. It's an every day fact of life for small to medium sized businesses.

Large organization often purchase document management systems that ultimately cost tens of thousands of dollars. Many of these products are aimed at vertical markets such as the finance industry, medical industry and so forth.

Document Management System Part 1	Document Management System Part 2	Document Management System Part 3
In this segment, we will; • Address Methodology • Create the database tables • Create the reusable 'Upload Wizard'	In this segment, we will; • Incorporate upload capability, storing the binary file to the database table. • Refine the 'Upload Wizard' • Create the menu to drive the interface.	In this segment, we will; • Create the form to allow users to allocate documents to specific users. • Allow the owner of the document to control the actions users can take with the document including 'check-in' and 'check-out' capability.

Unfortunately, it often doesn't matter whether you are a small to medium sized business - or for that matter a large organization… every organization needs or wants a centralized point where-in documents can be stored and the 'versions' of those documents 'controlled'. In addition, with today's Internet access, many organizations want their customers to be able to download the latest forms or documents and they want to do it as efficiently as possible.

The system we'll build in this series will enable you to provide your organization or users with these capabilities, along with the control processes they'll need to keep things current.

What Is A Document Management System?

There are many types of Document Management Systems out there. Each company producing these systems has their own definition depending on the features their application supports.

Many of the desktop versions provide hypertext capability, document imaging and other high end features. Much of the technology that enables these desktop systems to do the things they do involves COM/DLL components and are outside of the scope of this article.

For the purpose of this series, our fundamental document management system should enable authorized users to;

- Upload and download documents or binary files
- Control who can gain access to the documents, and
- Control who can modify or 'change' a document

By extrapolating this technology and molding around it a customized, logical framework you can build;

- a community based file exchange system,
- a family based web-site enabling family members to share photos, etc.
- hosted services, hosted download sites and so forth.

Therefore, as we go about the process of building the system we need to try to make it relatively foolproof to enable employees at any level of computer literacy to use the application with a minimum of training. Therefore, our system will use wizard interfaces where appropriate to ease the learning process.

Continued on page 12

CCS Components: The CCDLookup Component

Building A CCS Component, continued…

Continued from page 9…

Testing The Component

Now we're ready to test the code.

If you have Code Charge Studio running, close it down and then re-start CCS.

Open one of your projects if you like. We're going to test our component on a *new* page so that we won't be interfering with anything important.

Add a *new* blank page to your CCS. Save this page as "TestPage"

We're going to use this as our test page to ensure the code is being written as we expect.

Open the 'TestPage' page and add a BeforeShow event to it.

Then click on the HTML tab.

Place your cursor at the end of the page and hit Enter a few times to give yourself some room.

Now, place your cursor on the page and select the CCDLookUp component from your "Custom" tab.

The code shown in CCSC – Figure 5 will be written out to your HTML page.

Now, highlight the code highlighted in CCSC – Figure 5 and perform a cut, placing the code into your clipboard.

Delete what's left.

Now click to your events page, select the BeforeShow event and perform a paste.

Your "TestPage_events" should now look similar to that shown in CCSC – Figure 6.

If you've ever attempted to use a component within the _events page before, you probably received an error message that states "Toolbox is available only in Design or HTML mode".

This has not stopped us from utilizing the 'component' capabilities however. We simply have to use the bottom of the HTML page to get CCS to write it out, then do a little cutting and pasting.

In our next article, we're going to build a "Component Maker" that will simplify the process of creating our files and allow us to grab scripts from the internet and 'generate' components for those scripts.

This will also apply to our .asp, .php, .net, etc code.

While this was a simple example, we hope that you now have a much better concept of the Component building process and we hope that we've given you some tips along the way to helping you code more efficiently.

CCSC – Figure 5

```
<!--
 '********** ==> Routine Begins <== **********
'This routine is used to derive a value from a table in the database.
Dim sResult
Dim lngID
  lngID = Request.QueryString("id")
  If  Len(lngID) > 0 Then
    sResult = CCDLookup("FIELD_NAME", "TABLE_NAME", "id=" & CCToSQL(lngID,
"Integer"), DBIntranet)
  End If
FORMNAME.LABEL.value = sResult
'********** ==> Routine Ends <== **********
//-->
```

CCSC – Figure 6

```
Function Page_BeforeShow() 'Page_BeforeShow @1-653D685B

'Custom Code @2-73254650
' -------------------------
' Write your own code here.

Dim sResult
Dim lngID
        lngID = Request.QueryString("id")

        If  Len(lngID) > 0 Then
                sResult = CCDLookup("FIELD_NAME", "TABLE_NAME", "id=" & CCToSQL(lngID, "Integer"), DBIntranet)
        End If

        FORMNAME.LABEL.value = sResult

' -------------------------
'End Custom Code

End Function 'Close Page_BeforeShow @1-54C34B28
```

Building A Document Management System

Part 1 of a three part series, continued…

Creating The Database Structure

Such applications are best suited to the more powerful databases such as SQL Server, MySQL & Oracle.

Having said that, you'll be surprised at the ability of MS Access to handle a fairly good number of concurrent users.

So if MS Access is all you have, don't let that stop you.

In order to provide for the widest possible audience, we're going to build our application using MS Access.

It is highly recommended however, that if you are transferring the MS Access database to a high end database that you modify the SQL and code presented in this article to suit your database and that you create stored procedures (based on the SQL provided) to achieve the best possible performance from your system.

Before We Begin

Before we begin, we need to have a conception of a number of factors. Notwithstanding our need to have adequate programming skills to create the application, we also need to be concerned about how the user will interact with *the* system.

You *could* build the most sophisticated application ever, but *if* no one could operate it with efficiency or the system cannot be understood – it's worthless. So we need to build something that everyone can use without extensive training. Thus, we need to make it as intuitive or natural to the end-user as possible.

The 'system' we will build will be powerful, yet easy to use. It will also be extensible, meaning that you will be able to add additional tables to increase the functionality of the system to best suit your customers needs.

Don't be fooled by 'ease of use'. More often than not, an application that is powerful yet is easy to use takes more work on the part of the developer than most end users realize.

Creating The Tables

Included in the source code for this article is the DocManager.mdb file.

This file contains the base tables that we will start our project with. As we expand on the Document Manager in subsequent issues of CCS Developer Magazine, we'll expand on these tables and add new ones – increasing the functionality of our system.

Document Manager Tables Description
Users* Table Needed to ascertain the login parameters and the security level for a given user.
DocumentManagerDepartments Table Used to hold the names for various 'area's' or departments into which a user can assign a document.
DocumentManagerUploads Table Used to hold the binary files and information relating to the binary file.
DocumentManagerUploads2Users Table Used to hold the values necessary to assign documents to particular individuals.
* Note: The Users table used in this example is a cut-down version of the MS Access files that come in the CodeCharge Studio Example Pack. Therefore, when you upload the tables to SQL Server, MySQL or Oracle, etc – ensure that you do not overwrite *your* Users table.

For the moment however, the MS Access database file contains three (3) tables;

Users
DocumentManagerDepartments
DocumentManagerUploads

The source code for this article also contains a number of pre-built pages to allow us to get 'up and running' almost immediately.

Setting Up The Project

To set up a connection to the project, create a ODBC DSN named DocManager and point it to the project files .mdb directory.

For additional help on setting up an ODBC connection you should refer to the following on-line articles.

CodeCharge Studio KnowledgeBase article 54

Alternatively, you can set up a connection string to acquire access to the database.

In either event, you should be successful in ascertaining a connection to the database.

Creating The ODBC Connection / DSN

Create a DSN (Data Source Name) on our server so that we can establish an ODBC connection to the MS Access database.

Create a DSN on your server and call that DSN "DocManager".

Once you have your connection established, we're ready to begin.

You will also need to modify the connection string settings in the file dbUpload.asp. Open this file with your favorite HTML editor and modify the connection string there as well if necessary.

Security Settings

We have 5 levels of security with level five (5) being the administrator. We may not need this many, but as our document manager expands, it may be that we will.

If you are modifying this application to suit your special needs, change the levels accordingly.

Problems With This Method Of Security

Usually, there are no problems with this method for establishing the users access rights.

Building A Document Management System

Part 1 of a three part series, continued…

The problems generally arise when your application requires a dynamic method based on the page they're on or the document they're attempting to access.

CodeCharge Studio caters to so many of our needs that we find ourselves searching for the dynamic login method in CodeCharge Studio. But you won't find it in the previous or current versions.

That may be just as well, because there are a half a dozen ways to implement such a system and quite often, your application will dictate to a large extent – the method you need to use.

In this series of articles, we're going to use one of the methods to create a dynamic access capability when it comes to allowing users to upload, access and manipulate documents.

Creating A Re-usable Upload Wizard

One of the great things about CodeCharge Studio is the ability to point to a database and generate the forms and grids. We however, will not be doing that.

Because of the various levels of computer literacy one is likely to find in the workplace, we need to keep our application as simple as possible. Therefore, we will be making extensive use of Wizards in our web application.

By utilizing a wizard interface, we're going to keep users happy and try not to make more work for ourselves.

How?

There are a lot of benefits to using wizards in the more sophisticated areas of your application. If you use a wizard, the application can be made 'self-explanatory'. If you don't use a wizard for some things, the application can be confusing to the user and then we have to write help screens. Either way you're writing. You may as well write the wizard. Furthermore, as our application grows, we'll be able to insert additional functionality to the application by including that functionality as an addition page or sequence of pages within the wizard. This allows us to ask questions and channel the user to a common sense result. In other words, it allows us to build extendable - 'logical paths'.

Analysis Of The DocumentManagerUpload Table

The first thing we're going to do is break the table down into bite-sized chunks. We want the user to follow a process *dictated* by our wizard. We also want the user to focus only on one task at a time when they upload a document for the (*Insert*) first time. They'll likely have a slightly different path when they *update* the document.

Before we write any code however, lets open the database and open the DocumentManagerUpload table in design mode.

We know that "UserID" and "UploadDT" will be populated by the system, so we will create these as hidden fields. The two hidden fields will be present on the first page of the wizard. We're also going to add the "Title" and the "Description" field to the first page of the wizard. This provides a natural starting point for our Upload Wizard.

Remember, the files will be held in a binary format within the database, not physically located on the server's hard drive. We will need to be able to search for the document and the next field is called "keywords". This gives the user the opportunity to provide keywords used in the document. If you notice however, we've created this field as a Memo field.

The reason we did this is because it allows the user to literally paste word documents into the "keywords" field. If you scale this code up to SQL Server or Oracle, etc., you can create a "Full Text Index" on this field, thereby improving your search capabilities. By creating this field as a Memo field in our MS Access database, we've enhanced the likelihood that the correct document will be found during a search.

Next, the fields "Data, DataSize, ContentType and SourceFileName" will be handled for us by the upload component.

The file_area field will hold the value representing the department they want to allocate the file to.

In our next issue, we'll be extending the wizard to allow the user to allocate the file to one or more specific individuals.

And finally, we'll ask the user if they want to 'publish' the document.

By providing a "Publish?" check-box, we allow users to upload files into storage without that file being visible to any other users. This is good, because it allows a rough draft to be placed in storage for retrieval and updating later.

There's only one field left, and that's the 'times_downloaded' field. This field will be incremented whenever someone accesses the file. The code to do this therefore does not reside within the "Upload Wizard" and will be talked about later.

Building A Document Management System

Part 1 of a three part series, continued…

Even though we're using the "Upload Wizard" in a document management system in this article, you should separate this wizard in your mind from the rest of the application. The "Upload Wizard" is a write once - use many times application in it's own right.

The "Upload Wizards" purpose is to allow users to upload files and it knows nothing about the system it's plugged in to. Tomorrow you might have to build a family picture sharing web-site and the "Upload Wizard" will be there for you to use virtually without alteration, saving you time and allowing you to focus on the surrounding application instead.

If later, you need to add more fields to the DocumentManagerUpload table, you simply insert another page/step into the upload wizard to accommodate your requirements.

Building The Reusable Upload Wizard

There is no trick to building wizard interfaces really. However to build them you do need to make sure that the table allows a record to be created with only one or more fields to be filled in at a time. In other words, all fields *except* the critical ones - *must* accept NULL values. If you have any fields that do not allow NULL fields, then those fields need to be on the first page of the upload wizard.

For our DocumentManagerUpload table to allow a wizard interface, all fields will be NULL except for the "UserID, UploadDT and Title" field.

Since the "Title" field must have a value to insert the record, that field is appropriately on the first page of our wizard.

We also need to capture the Users ID and the current date/time within hidden fields. This is important, because when the user clicks over to page 2 of the Wizard, we're going to check that the User_ID in the table matches the session('user_id") for the record the user just inserted and whether the document id being passed in the querystring is owned by the user.

If we didn't do this, anyone could simply modify the UploadID passed in the URL/Querystring and gain access to some one else's file. This would not be good.

If someone comes along afterwards and plays with the querystring values we'll send the user to our AccessDenied page.

Building The Upload Wizard Entry Point

When the user fills in the "Title" field and clicks submit, an *insert* is performed. Every page of the wizard after the first page performs an *update* to the record. Therefore, we need to add some code in the After_Insert event to derive the Autonumber or IDENTITY field. We will then pass the Autonumber/IDENTITY field to the next and subsequent pages.

If the record id *does not* get passed to the subsequent pages, the update (submit) button will *not* be visible. This is generally a clear indication that the record id – in this case "UploadID" was not passed to the page.

Open the DocumentManagerTitle.ccp file and navigate to the After_Insert event for the record.

You'll notice that the code required to get the Auto-number from an MS Access database. We've also added the code to derive the @@IDENTITY from MS SQL Server. The CodeCharge Studio help file (F1) demonstrates these as well as examples for databases other than MS Access and MS SQL Server.

Open the default.ccp file. You'll notice that we've only placed the critical fields on the first page of the wizard. Refer to figure DMS-1.

The user has to enter the title and description of the file. The user_id will be saved in a hidden field as well as the date the file was uploaded.

Denying Access

Since we need the userid, the user must be logged in to access the page.

If they're not logged in, we need to tell the user that they need to login in order to access the page. Therefore we're going to set security for the page and we need to have an Access Denied form to let the user know what's going on.

Open the AccessDenied page. The user will be directed to this page if they have not acquired a session via the login.

While we're at it, open the InsufficientAccess page. This page will be used to advise the user in the event that they *are* logged in, but their access rights are not sufficient to perform the action they attempted.

This will be important later when we build the document management logic into the application.

Refer to figure DMS-2 for the datasource Where clause settings.

Allowing The User To Add Keywords

We need to allow users to search for the documents held in a binary format in the "Data" field.

This page of the wizard allows the user to add keywords and even full documents.

In the case of an MS Word document for instance, the user *could* copy and paste the entire document into the field.

In the case of an image file being uploaded, the user may choose only to add a ten or twenty word explanation.

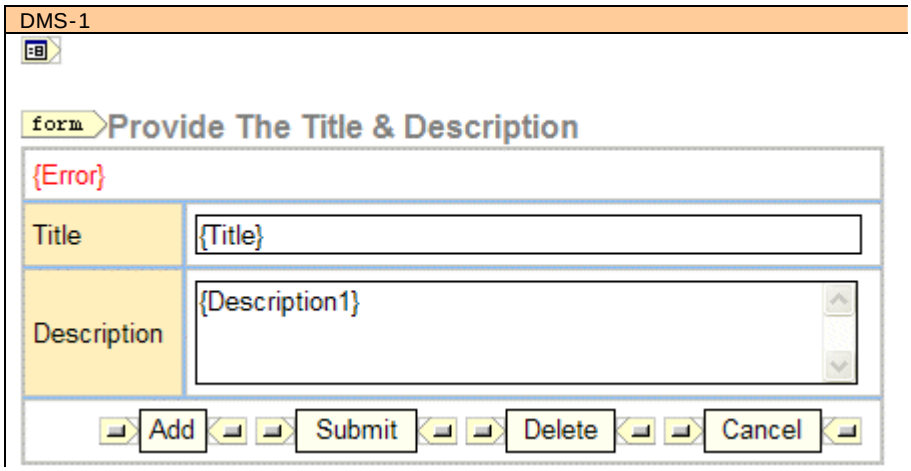
Step 2 of the wizard therefore involves providing the user with the opportunity to make the document or other binary file – 'searchable'.

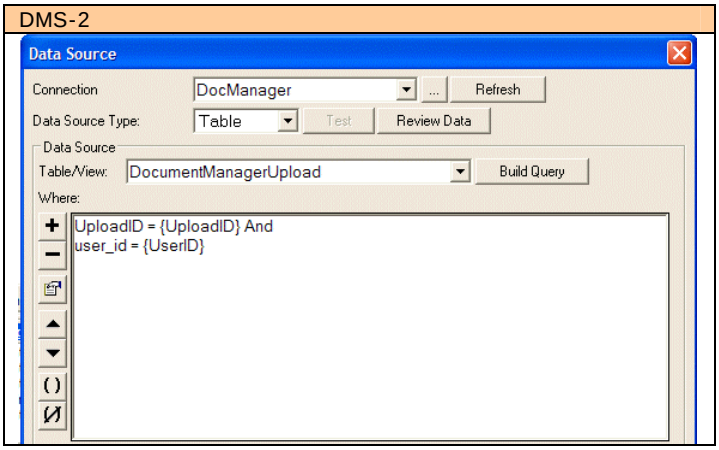
By maintaining the user to using a 'TextArea' field, we prevent rich text and embedded images from being placed within the 'Keywords' field.

If you need this capability in your application you can incorporate a rich text capable web based editor such as FCKEditor in this step of the wizard.

Building A Document Management System

Part 1 of a three part series, continued…





Adding The Rich Text FCKEditor

The FCKEditor can be found at http://www.fckeditor.net/

There is a new version in Beta release at the moment, so we're going to plug in an earlier version in this article. When the new version comes out of Beta mode, we'll show you how to integrate the new version in a future article.

If you need more assistance with the FCKEditor you should visit their web site for documentation and additional examples.

This wizard is going to use a Frame to act as a container for the WYSIWYG Editor.

There are two (2) ways we can plug the frame into our page.

We can

- write iFrame code directly into the html page, or
- we can add a label and use the before show event to produce the iFrame code.

In this article we're going to demonstrate the technique of using a label to produce code and display it.

This technique can be used for a lot of different scenarios, so it's important to know how to do it (it's easy).

Sometimes, if you paste certain code into the CodeCharge Studio HTML editor, it will disrupt the carriage returns and everything will end up in one long line of code.

You can also place your code between <!-- --> tags and this will usually keep the sentences formatted as you'd expect.

But there are also more subtle reasons for using the label field in this way.

This technique also allows you to programmatically create an iFrame or some other object according to user rights. But we'll address that in a future article.

Click on the {Description1} text area, perform a right click and select "Change To" → Hidden. Move the Hidden Description1 field out of the way. We placed it after the Title Field.

Next add a label field and give it the name lblWYSIWYG. While still on the label, click the events tab, perform a right click and "Add Code" to the Before_Show() event.

Take a look at the code that we added to the DocumentManager_Default_events.asp page for the Before_Show() event.

Essentially all the code does is replicate what we'd otherwise type into the HTML page and assign it to the label.

The FCKEditor directory is located as a sub-directory of the DocumentManager Directory and loads into the iFrame accordingly.

If you decide to copy and paste this routine into applications you're writing with CodeCharge Studio, you only need to alter the FieldName="YOURFIELDNAME" portion of the line and perhaps the Browse and Upload Boolean values.

[?FieldName=Keywords&Upload=true&Browse=true]

That's all there is to adding the FCKEditor to your MEMO and TEXT fields.

Uploading The File To The Database

Building A Document Management System

Part 1 of a three part series, continued…

There are two (2) ways for using the non-COM based file upload component that ships with CodeCharge Studio.

One methods allows you to perform a script based upload and the other method allows you to perform an ADO.Stream based file upload.

In both cases it appears that the component requires your server to have a temporary directory into which the file is initially loaded and a 'permanent' directory into which the file is subsequently moved if it passes validation.

However, there are times when the application will require that the Upload component must upload the file directly into a field in a database table. The reasons can range from the need for additional security or simply because the ISP that hosts your site is unwilling to allow sufficient permissions to the necessary directories.

Furthermore, if you do have sufficient permissions on these directories, you may be concerned that the directories could be discovered and in one way or another compromised.

As a result of this concern, you have probably adjusted your component to disallow the uploading of certain types of files, especially .exe, .com, .bat files and so forth. And you would be correct to have done so.

Still, there will be times when a system requires a higher level of security or the ability to upload the executable type files. And thus make it harder for a hacker to compromise your web sites security measures.

Uploading the file directly to the database and downloading the file directly from the database may by your only choice.

Since CodeCharge Studio comes with sufficient information about their upload component, our application is going to use a different one.

The component we're going to use is a free Upload component called PureASPUpload. This component is apparently capable of uploading files up to 2 GIG in size.

Normally however, this limitation is no limitation at all.

This component can be downloaded from http://www.ormacdigital.com/pureaspupload/help/default.htm

They also sell another product called HUGE-ASP Upload that allows you to upload files larger then 2 GIG.

Note: Downloading documents, images and other binary files from your database can cause a strain on both the database and your bandwidth. Unfortunately, reality often dictates things and we have no choice but to perform uploads directly to a table.

Since this component allows files to be uploaded directly to a database field and is free, it's a likely choice to use until the CodeCharge Studio component provides similar functionality.

Using PureASP Upload

All of the necessary files have been include along with the CCS Source code for this article.

When you download the PureASP Upload zip file, the pureupload.asp file and others are contained inside. Therefore, it is recommended that you do not modify the pureupload.asp file.

We chopped down/modified one of the example files and renamed it to dbUpload.asp

Using your favorite HTML editor, open the dbUpload.asp file.

The dbUpload.asp file references an include: `<!--#INCLUDE FILE="./pureupload.asp"-->`

Again, there is no need to modify the pureupload.asp file.

In this issue we're not going to restrict the types of files we can upload. Since the files are being uploaded to a table within a database we can be less restrictive.

We should however limit the size of the file that can be uploaded. Located at or about line 35 you'll see two (2) lines of code that you will need to modify if you want to alter the allowable file size.

```
Server.ScriptTimeout = 10000
Form.SizeLimit = 1024*1024*2
        '1MB=1024*1024
```

We've set the file size limit to 2MB.

As you increase the file size you may also need to increase the Server.ScriptTimeout value.

We've set the Server.ScriptTimeout a little higher than one might expect to see.

Adjust this setting according to your needs. Just remember that if some or many of your users access your site using a 56k dial-up connection, you'll need to keep the Time out set to accommodate them – or their session will time out part-way through the upload.

We've also taken the liberty to simplify the set-up required to integrate these files into your existing applications.

At the top of the dbUpload.asp page, you'll find a few variables that you may need to alter should you transfer these files to one of your own projects.

Deleting The File Without Deleting The Record

To delete the file without deleting the entire record, we're going to create a separate page.

Assigning The Department

After the file is uploaded, the wizard takes our user to the page that allows the user to assign the file to a department.

The department could represent an actual department of an organization or it could represent a category.

What-ever it represents within the context of your application, name it accordingly.

If you don't need to break the file down in this fashion, simply remove this step from the wizard.

After assigning the file to a department, the user is take to

Building A Document Management System

Part 1 of a three part series, continued…

the List Of Files page.

The List Of Files Page

We didn't include the keywords field on the grid because if a user copied and pasted an 18 page report into the system, the grid would scroll for a long way indeed.

Instead, we've added the fields that you're most interested in seeing populated by the wizard.

We also created an additional page to handle the various types of actions that a user can take with the document. As our article progresses in future articles this capability will be enhanced and the list of options will grow.

Wrapping Up

At this point in our application we have addressed the core functionality required to upload files to our database.

In our next issue, we'll be expanding on both the number of options the user will have as well as implementing the dynamic 'rights' each user will have for a given document.

We've demonstrated how to use the FCKEditor to add rich text capabilities to your application and we've demonstrated a number of techniques that we'll use to good effect as we continue building the application.