

SoftWidgets™ Spread Server 3.0 Reference Implementation

Copyright © 1999-2006 SoftWidgets Corporation. All Rights Reserved.
Document Version: 5. Last updated: 1/10/2006.

Contents

[Introduction](#)

[Description of the Hypothetical Business Requirements](#)

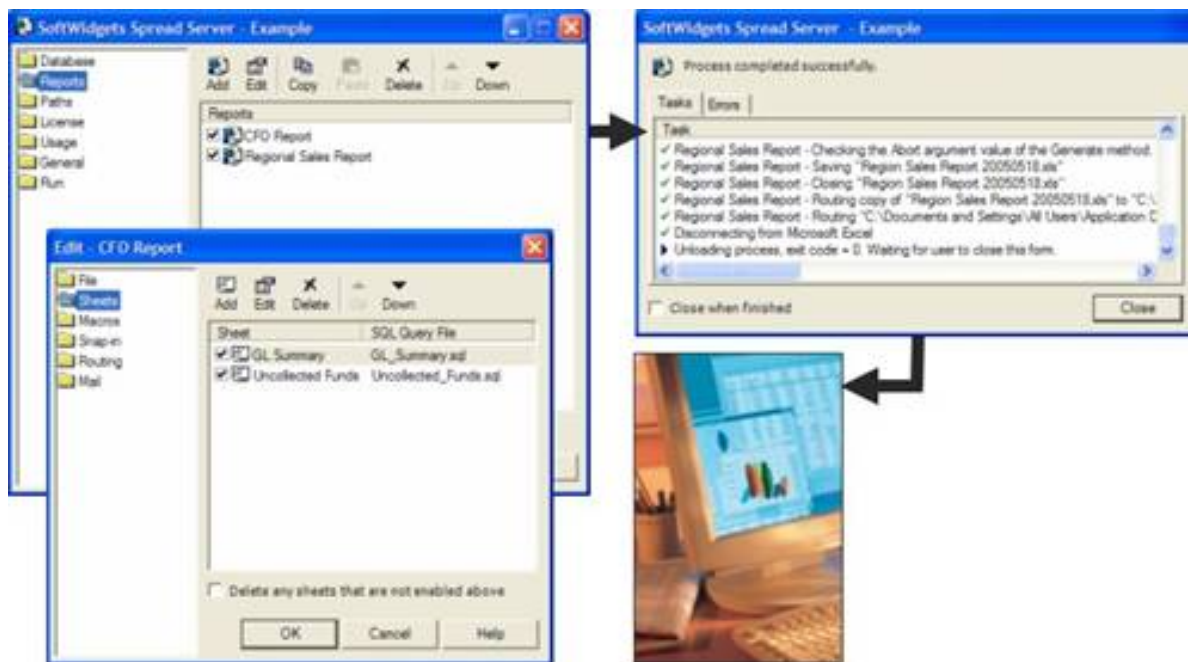
[Implementing the Solution using a Simple SQL Statement](#)

[Implementing the Solution using a .NET Snap-In](#)

[Conclusion](#)

Introduction

As illustrated below, this document describes how to implement a SoftWidgets Spread Server solution to create a business user friendly Excel-based reporting system.



The document is intended to serve as a reference implementation for business analysts, Excel power users, software developers or IT decision makers who are building-out or seeking to improve upon their existing Business Intelligence and/or reporting capabilities.

Description of the Hypothetical Business Requirements

The top executives of “AnyCompany” would like to have a summary-level report of their total and monthly sales numbers broken out by division. Also, they want the report to be nicely formatted and delivered as an Excel file attachment in their email inboxes every morning, no later than 7:00 AM.

Implementing the Solution using a Simple SQL Statement

The cookbook approach herein describes the method for implementing the “AnyCompany” reporting solution using a simple SQL statement.

NOTE: If you wish to follow along with these implementation steps, in a tutorial mode on your own computer, then...

1. [Download the 30 day free trial of Spread Server](#) and install it on your computer.
2. [Download the reference implementation zip file](#) that contains all the files cited in this document.

NOTE: If you are having trouble downloading the above zip file from Acrobat, try typing or copying and pasting the link below directly into the address field of a new browser window and then press the enter key.

http://softwidgets.com/downloads/Spread_Server_Reference_Implementation_0300-01.zip

The implementation steps are as follows:

1. As seen below, create a template that meets the report look and feel requirements defined by the “AnyCompany” business users.

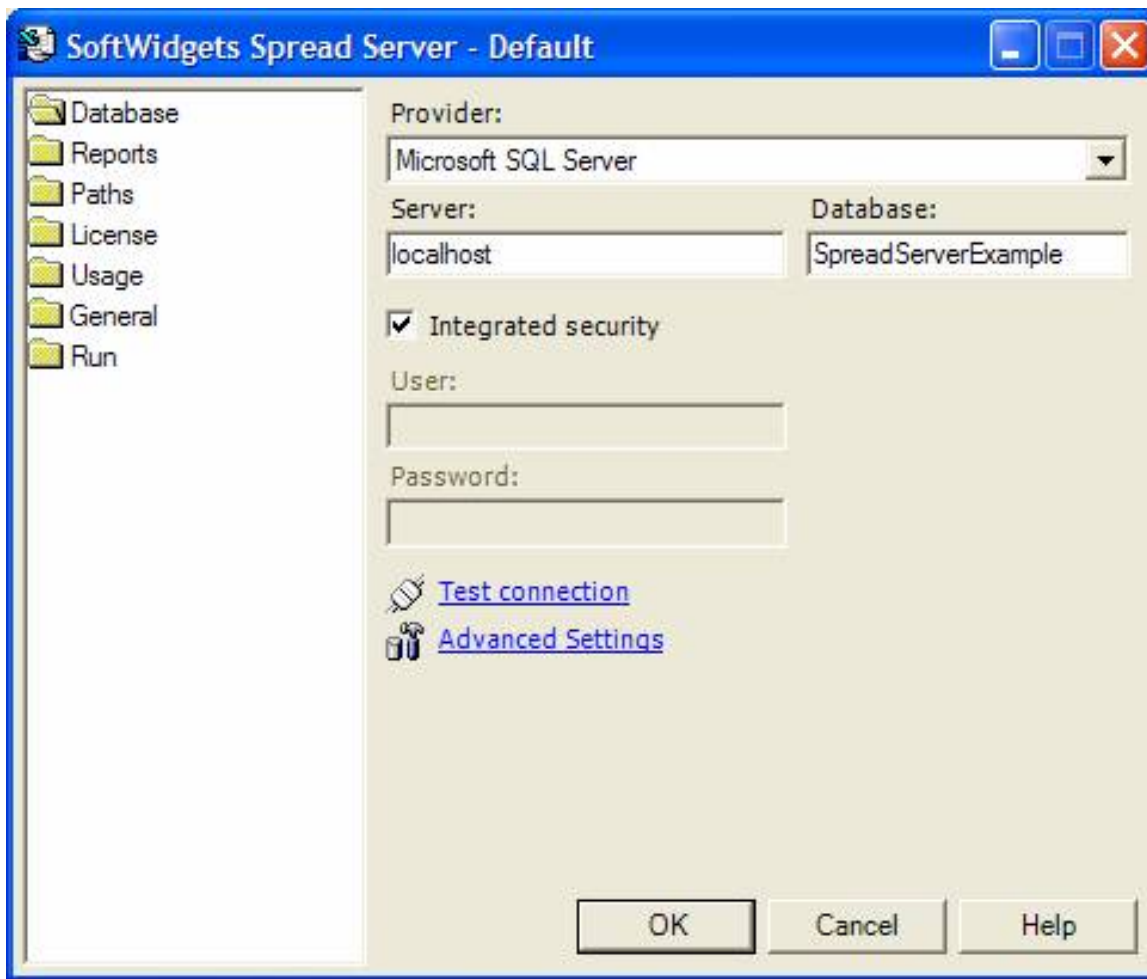
NOTE: A copy of the template named “ExampleTemplate.xls”, shown below, is provided in the zip file that accompanies this document.

The screenshot shows a Microsoft Excel window with the following data:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Division	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
2	Eastern													0
3	Western													0
4	Total	0	0	0	0	0	0	0	0	0	0	0	0	0
5														

2. From your “Programs\SoftWidgets\Spread Server 3.0” Windows start menu, click on the Spread Server menu item.
3. As seen below, the Spread Server configuration editor will appear.

NOTE: If you do not have a licensed copy of Spread Server, a screen prompting you to evaluate or request a Spread Server license will appear. You can just click on the “Evaluate...” link until you decide that you want to acquire license.



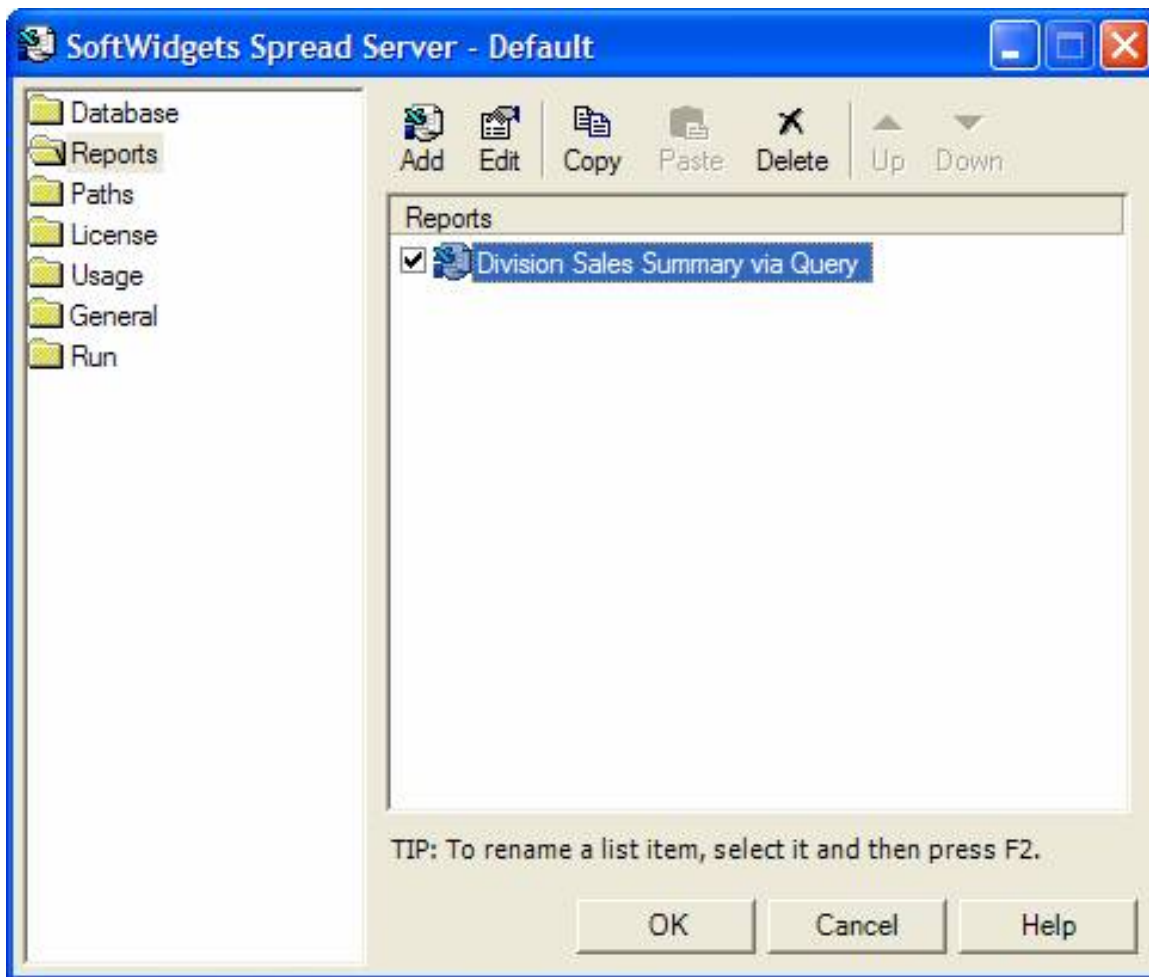
4. As seen above, configure the properties for the database from which the sales data will be obtained (i.e. this is where your sales data is going to come from). For the purposes of this example, a local instance of a SQL Server database named “SpreadServerExample” is specified and Integrated Security is checked. If you prefer, you can use another type of database (e.g. Oracle, DB2, Access, etc.).

To follow along with this tutorial, you will need to complete a few quick database setup prerequisites. A SQL script named “ExampleStructure.sql”, that defines the “SpreadServerExample” database structure and an Access database named “ExampleData.mdb” that contains the data used for this example are included in the zip file that accompanies this document. The database setup steps are as follows:

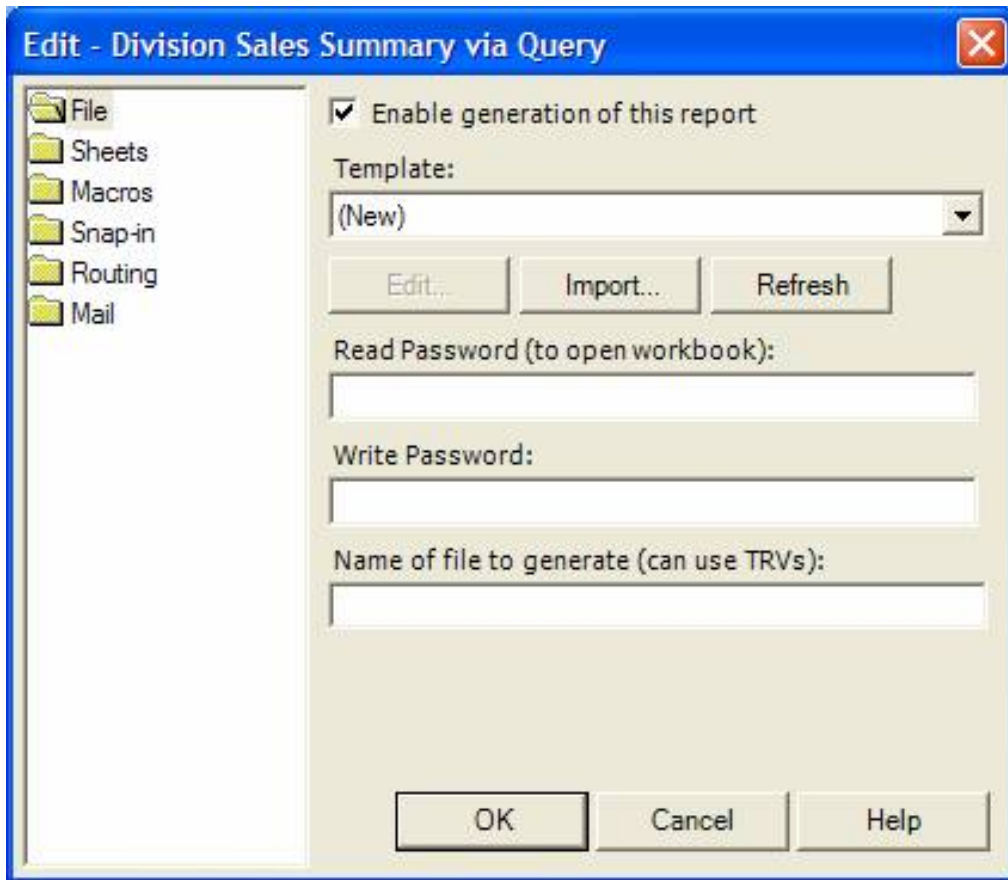
- 1) Create a SQL Server database or Oracle schema or DB2 table space, etc., called “SpreadServerExample”.
- 2) Run the script provided in the “ExampleStructure.sql” file to create the DivisionRevenue table.
- 3) Use DTS, SQL Loader, etc., to load the sample Division Revenue data from the “ExampleData.mdb” file into the DivisionRevenue table.

If you do not have a SQL Server, Oracle or DB2 instance available, than you can specify the “Other” in the “Provider” field and configure Spread Server to obtain the data from the Access database that is contained in the zip file that accompanies this document. The “Other” provider option allows you to connect to any ODBC-compliant data source with a connection string. Click on the Spread Server help button for an example of a connection string, if necessary.

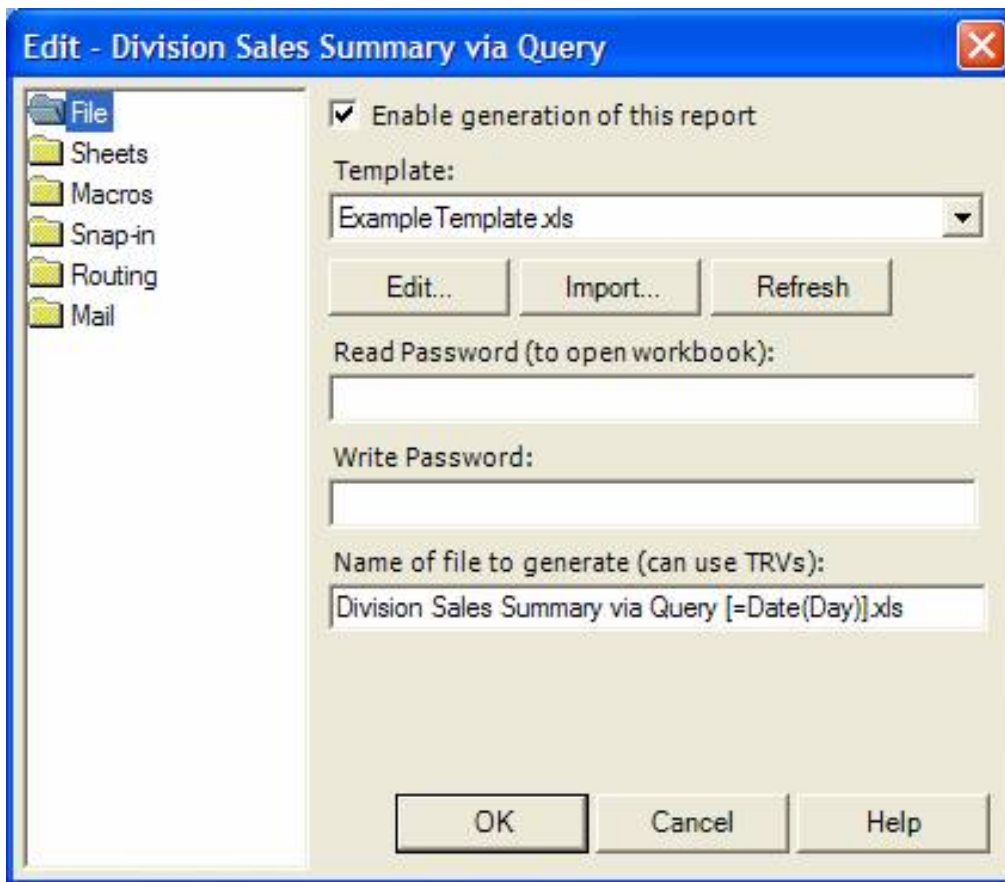
5. As seen below, go to the “Reports” properties and click on the “Add” icon to add a report. For the purposes of this example, name the report “Division Sales Summary via Query”.



6. Click on the “Edit” button seen above and a report editor should appear as seen below.

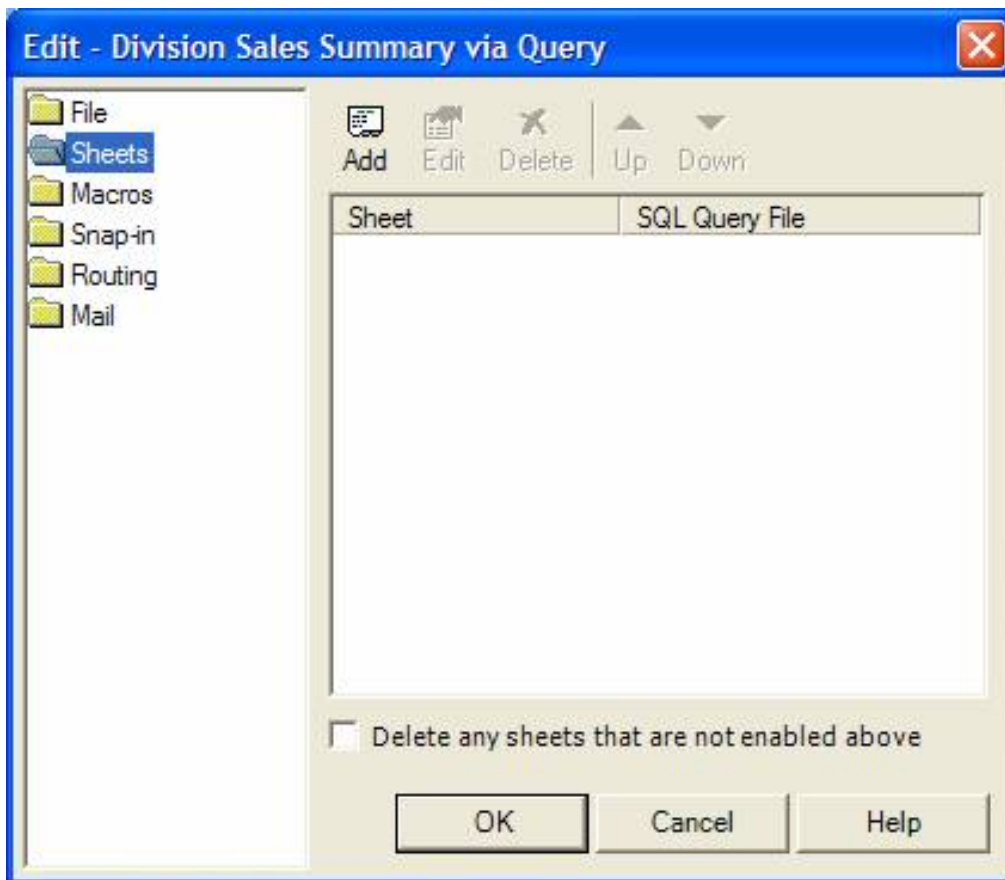


7. Click on the “Import” button shown above to import the template that was defined in step 1. A copy of the template named “ExampleTemplate.xls” is included in the zip file that accompanies this document.



8. As seen above, specify a file name by typing in “Division Sales Summary via Query [=Date(Day)].xls” in the “Name of file to generate” field. This is the file name as the end users will see it on a file share or in their inbox. The “[=Date(Day)]” token in the file name allows for a powerful dynamic run-time naming feature. In this case, the fully padded day, in YYYYMMDD format, will be included in the file name to make it easier for your users to manage the spreadsheets that are sent to them every day. The token conventions are fully documented and explained in the Spread Server help files.

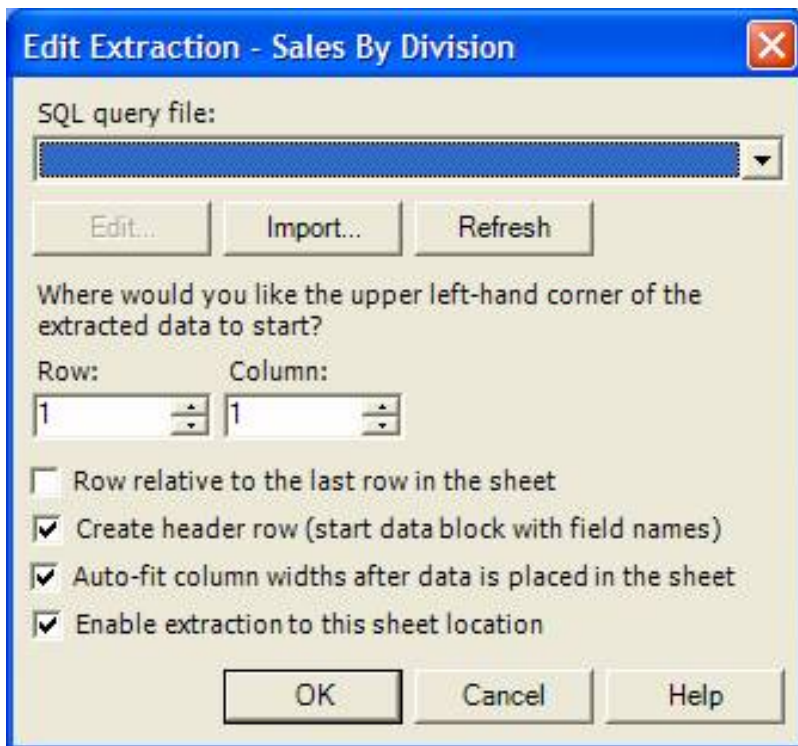
9. As seen below, click on the “Sheets” properties folder.



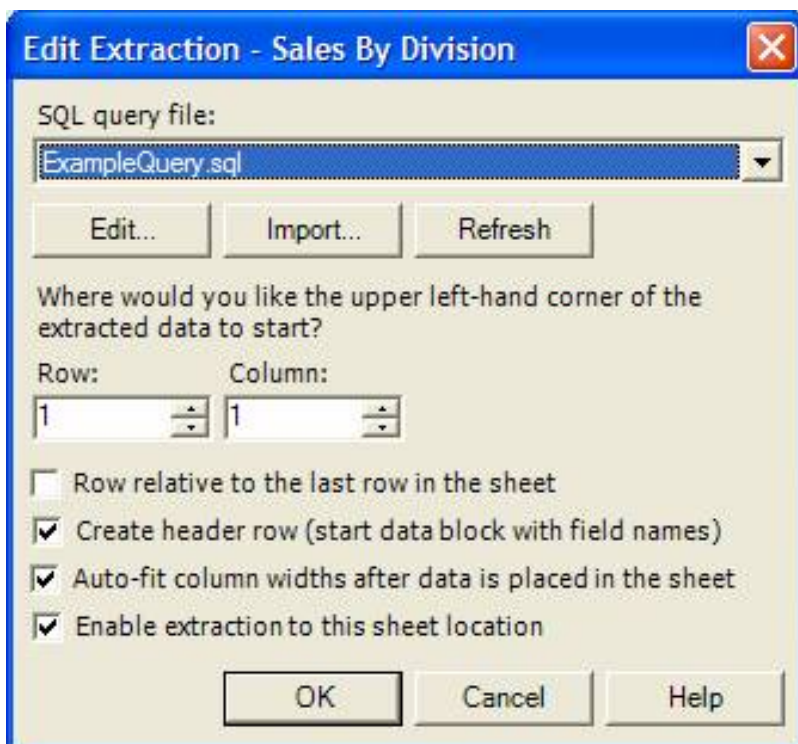
10. Click on the “Add” icon to add a new sheet. Name the sheet “Sales By Division” as seen below.



11. Click on the “Edit” button and an extraction editor will appear as seen below.



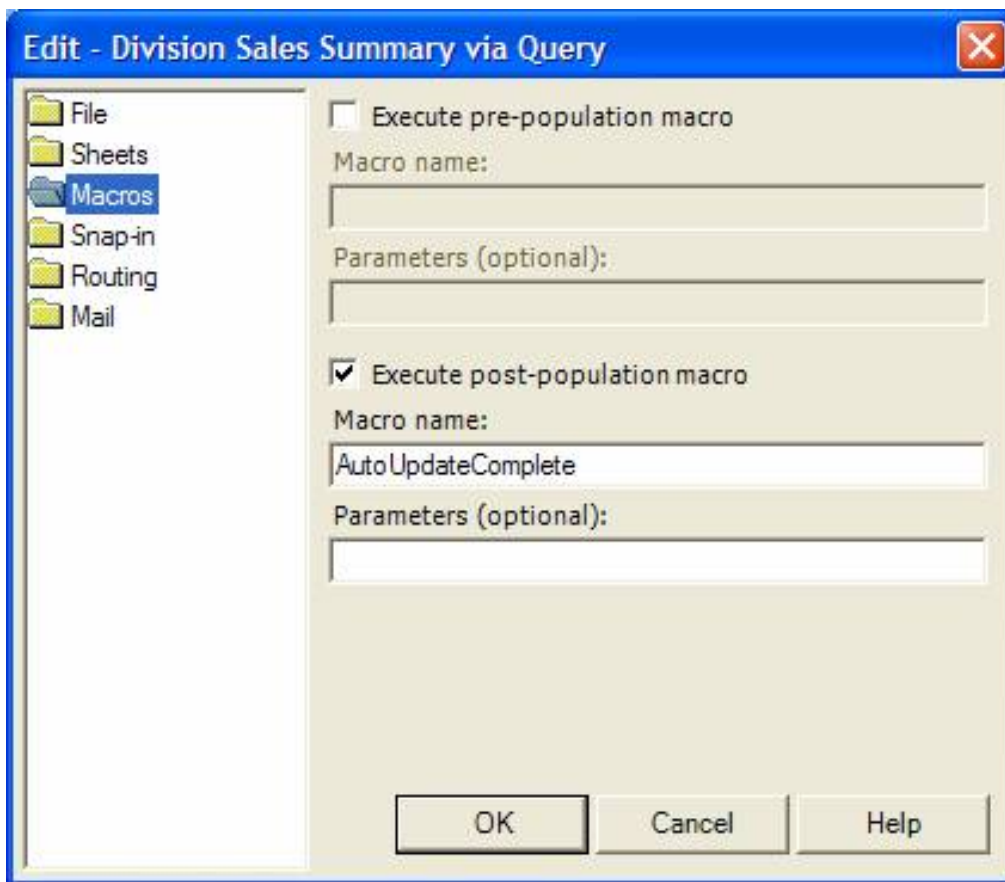
12. Click on the “Import SQL query file” link to import the “ExampleQuery.sql” file that is contained in the zip file that accompanies this document. As seen below, after the import of the SQL file, “ExampleQuery.sql” will be selected in the drop down.



NOTE: If you are following along using Access as your database then you should specify the “ExampleQueryForAccessOnly.sql” query in the above drop down. This Access query is simplified so that you can complete this tutorial using Access. Attempting to use the “ExampleQuery.sql” query with Access will not work because Access does not support composite queries (i.e. queries with multiple statements). The more advanced “ExampleQuery.sql” query used in this example is more representative of a real-world scenario where you would be manipulating normalized data in an enterprise-class database environment before returning results to Spread Server.

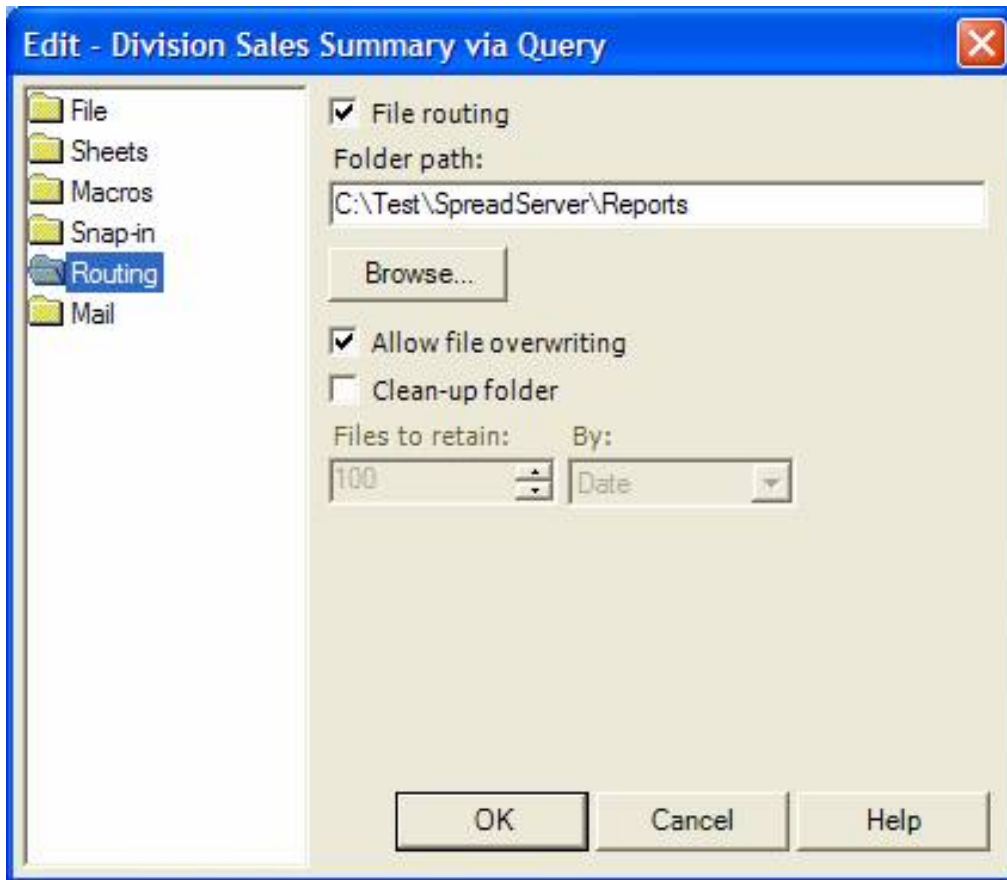
13. Click the OK button.

14. To demonstrate the powerful new macro execution features, click on the “Macros” properties folder as seen below.



Configure as is above and a macro named “AutoUpdateComplete” that is located in the “ExampleTemplate.xls” file will be executed after the data population has completed. You can check out the macro in the template to see that it will insert a last update stamp at the bottom of the sheet.

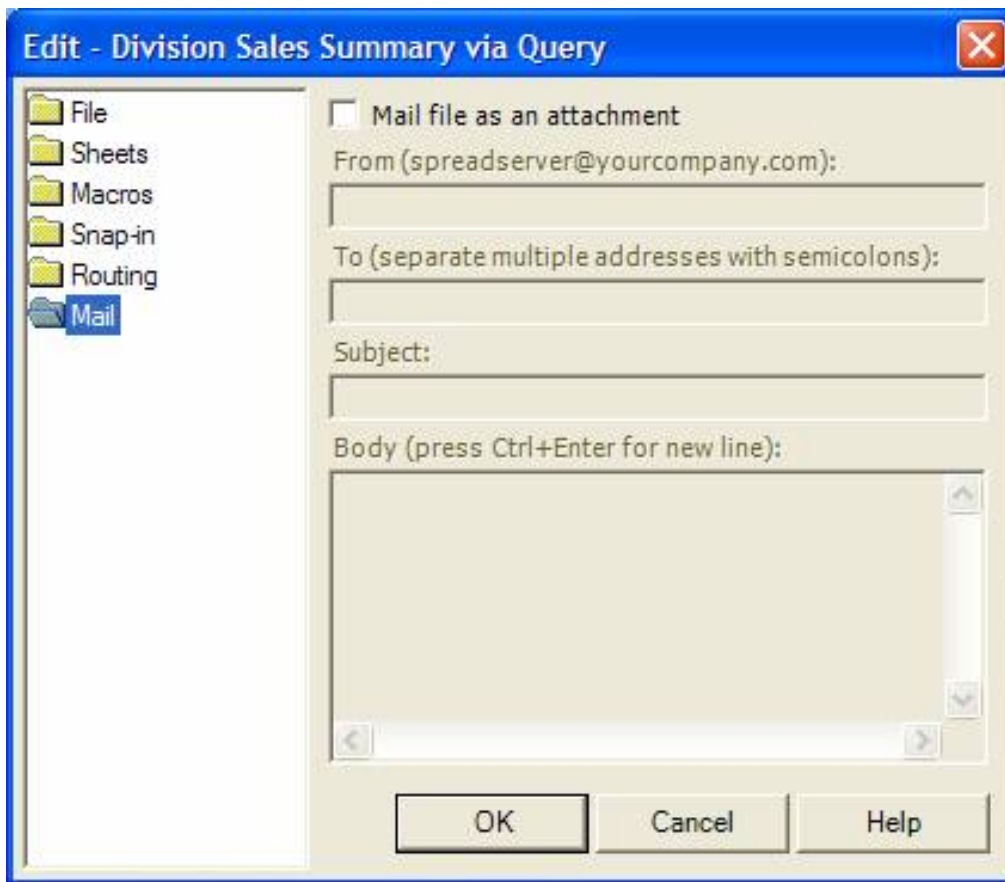
15. Click on the “Routing” properties folder and configure as seen below.



NOTE: If the path specified in the “Folder path” field shown above does not exist, Spread Server will automatically create the entire path for you at run-time.

Notice, also, that for the purposes of this example, “Allow file overwriting” is also checked. Since your files are named uniquely on a per day basis, this setting will prevent a “cannot overwrite” error if you run this example more than once in the same day.

The business requirements call for delivery via email as opposed to routing to a folder as we have configured above. Email routing can very easily be accomplished by clicking on the “Mail” folder seen below and configuring the properties as desired.

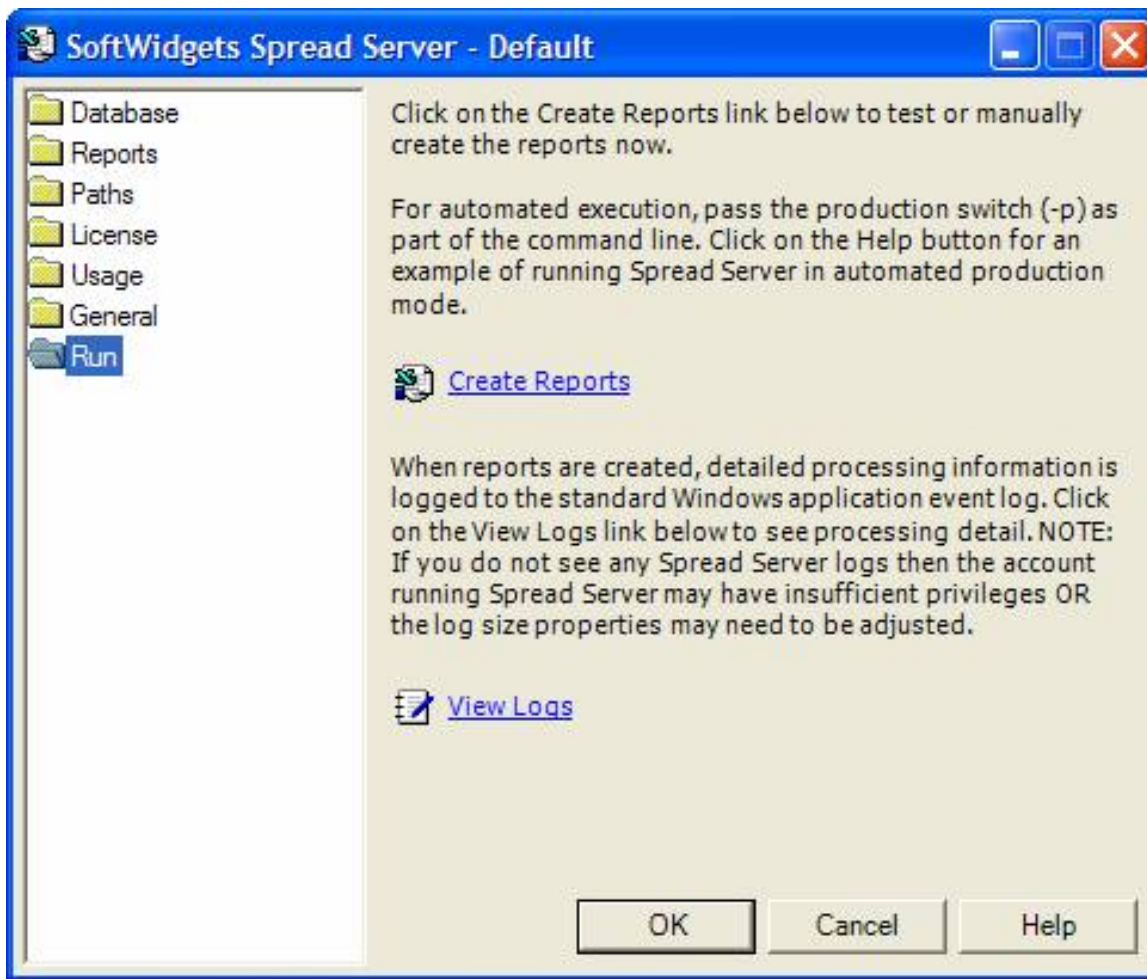


For the purposes of this example and since it is not known if the reader of this document has the Windows SMTP service installed, the generated files will not be emailed but they will be routed to “C:\Test\SpreadServer\Reports” on the local hard drive instead.

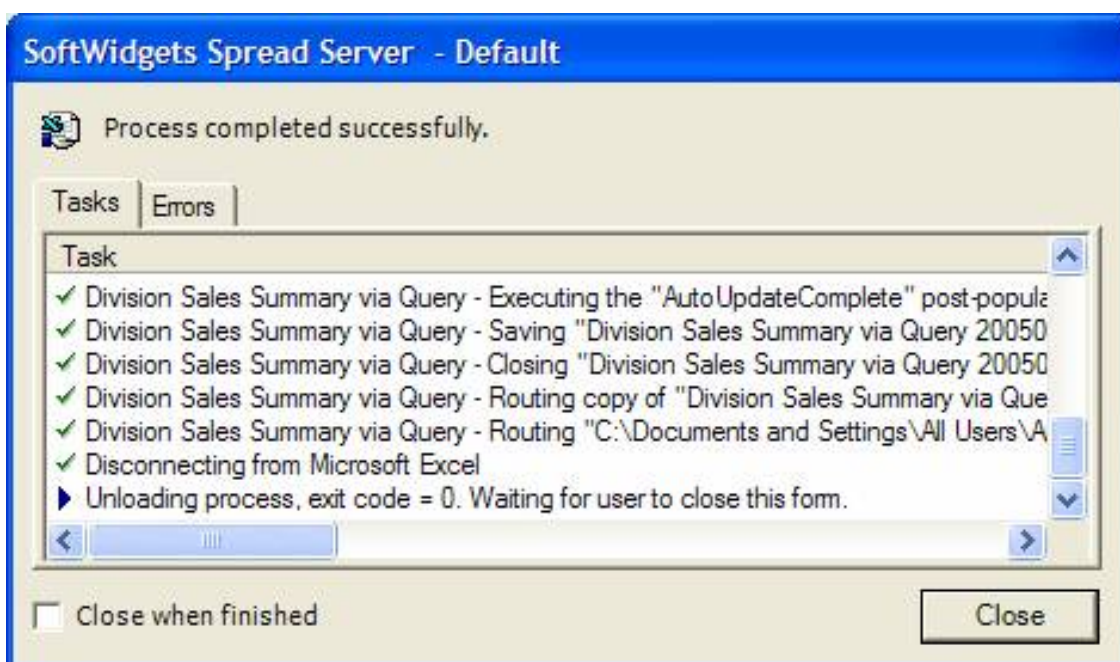
16. Click OK. The main Spread Server editing screen will receive the focus.

This completes the Spread Server configuration steps and we are now ready to run the process to generate the spreadsheet.

17. As seen below, click on the “Run” properties folder.



18. As seen on the screen above, click on the “Create Reports” link to begin the spreadsheet generation process. Confirm the run request and a status monitoring window, like the one below, will appear as Spread Server is creating the reports.

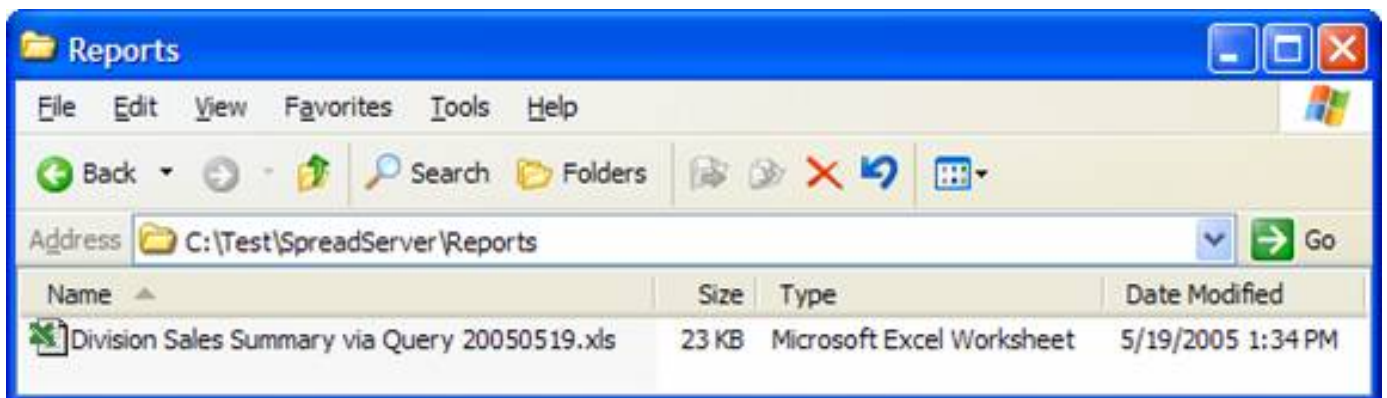


If something goes wrong during the spreadsheet generation you will see a task failure as indicated by a yellow circle containing an exclamation. You can determine the nature and the cause of the failure by clicking on the Errors tab or by examining the standard Windows application event logs.

NOTE: This monitoring interface only appears when you execute a manual/test run from the Spread Server configuration editor. It DOES NOT appear in production mode where, in accordance with server-side application best practices, no UI is presented during execution. In production mode, you must review the standard Windows application event logs to ascertain Spread Server's execution status and history.

19. Once the processing is complete, as seen above, close the status monitor and you will be returned to the Spread Server configuration editor window.

20. As seen below, you can examine the results of the spreadsheet generation job, once it has finished, by opening the generated spreadsheet in the folder where it was routed.



When opened, the spreadsheet should look as follows:

The screenshot shows Microsoft Excel with the following data:

Division	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
Eastern	100,987	125,000	97,050	-2,300	4,056	87,544	130,021	-455	10	122,677	159,733	101,740	926,063
Western	200,004	300,232	-22,300	-11,341	594,002	22,000	76,451	-17,455	90,801	95,788	100,452	257,912	1,686,546
Total	300,991	425,232	74,750	-13,641	598,058	109,544	206,472	-17,910	90,811	218,465	260,185	359,652	2,612,609

Additional information from the screenshot: The status bar shows 'Sales By Division/' and 'Ready'. The bottom right corner displays 'NUM'.

Notice that the "AutoUpdateComplete" post-population macro put the "Last

auto-update” stamp in the sheet as show above in green.

Implementing the Solution using a .NET Snap-In

Spread Server snap-in implementations are alternatives to the simple SQL statement based implementation that was discussed in the previous section. Snap-in implementations are appropriate when you have more complex spreadsheet generation requirements that cannot be accommodated without extending Spread Server’s out-of-box functionality.

Developing a snap-in for Spread Server is a straight forward interface based approach to application extensibility. Below are brief descriptions of the interfaces that are supported by Spread Server:

Interface	Description
SpreadServer.Interfaces.IBypass	Allows the generation of a report to be dynamically bypassed at run time.
SpreadServer.Interfaces.ITemplateNamer	Allows a template name to be dynamically altered or specified at run time.
SpreadServer.Interfaces.IFileNamer	Allows a file name to be dynamically altered or specified at run time.
SpreadServer.Interfaces.IQuery	Allows a SQL query to be dynamically altered or specified at run time.
SpreadServer.Interfaces.IGenerator	Allows the generation of the spreadsheet to be delegated / handled at run time.
SpreadServer.Interfaces.IRouter	Allows a routing destination (folder path) to be dynamically altered or specified at run time.
SpreadServer.Interfaces.IMailer	Allows email message properties to be dynamically altered or specified at run time.

Below are simplified code samples in C# and VB.NET that demonstrate how to implement the interfaces listed above.

C# Spread Server Snap-in Interface Examples:

```

using System;
using Excel = Microsoft.Office.Interop.Excel;
namespace SpreadServerSnapInExampleCS
{
    public class Example :
        SpreadServer.Interfaces.IBypass,
        SpreadServer.Interfaces.ITemplateNamer,
        SpreadServer.Interfaces.IFileNamer,
        SpreadServer.Interfaces.IQuery,
        SpreadServer.Interfaces.IGenerator,
        SpreadServer.Interfaces.IRouter,
        SpreadServer.Interfaces.IMailer
    {
        public Example()
        {
        }

        bool SpreadServer.Interfaces.IBypass.Bypass
            (System.Data.IDbConnection Connection,
             string ReportName, string Parameters)
        {
            //For whatever reason, bypass if it is a leap year.
            if (System.DateTime.DaysInMonth(2003, 12) == 30)
                return true;
            else
                return false;
        }

        string SpreadServer.Interfaces.ITemplateNamer.FullName
            (System.Data.IDbConnection Connection,
             string ReportName, string Current, string Parameters)
        {
            return "C:\\MyCustomTemplates\\MyCustomTemplate.xls";
        }

        string SpreadServer.Interfaces.IFileNamer.ShortName
            (System.Data.IDbConnection Connection,
             string ReportName, string Current, string Parameters)
        {
            return "MyCustomFileName.xls";
        }

        string SpreadServer.Interfaces.IQuery.Query
            (System.Data.IDbConnection Connection,
             string ReportName,
             string SheetName, string Current, string Parameters)
        {
            return "SELECT * FROM SOME_TABLE";
        }

        void SpreadServer.Interfaces.IGenerator.Generate
            (System.Data.IDbConnection Connection,
             string ReportName, Excel.Workbook Workbook,
             ref bool Abort, string Parameters)
    }
}

```

```

{
    Excel.Worksheet Worksheet;
    Worksheet = (Excel.Worksheet)Workbook.Worksheets[1];
    Worksheet.Cells[1, 1] = "My custom spreadsheet generation.";
}

string SpreadServer.Interfaces.IRouter.Destination
(System.Data.IDbConnection Connection,
 string ReportName, string Current, string Parameters)
{
    return "C:\\MyCustomDestinationFolder";
}

void SpreadServer.Interfaces.IMailer.Compose
(System.Data.IDbConnection Connection,
 string ReportName, System.Web.Mail.MailMessage Message,
 string Parameters)
{
    //NOTE: All unassigned Message properties default
    //to the values as configured in Spread Server.
    Message.Subject = "My custom email subject";
}
}
}

```

VB.NET Spread Server Snap-in Interface Examples:

```

Imports Excel = Microsoft.Office.Interop.Excel

Public Class Example
    Implements SpreadServer.Interfaces.IBypass
    Implements SpreadServer.Interfaces.ITemplateNamer
    Implements SpreadServer.Interfaces.IFileNamer
    Implements SpreadServer.Interfaces.IQuery
    Implements SpreadServer.Interfaces.IGenerator
    Implements SpreadServer.Interfaces.IRouter
    Implements SpreadServer.Interfaces.IMailer

    Private Function Bypass _
        (ByVal Connection As System.Data.IDbConnection, _
        ByVal ReportName As String, _
        ByVal Parameters As String) As Boolean _
        Implements SpreadServer.Interfaces.IBypass.Bypass

        'For whatever reason, bypass if it is a leap year.

        If System.DateTime.DaysInMonth(2003, 12) = 30 Then
            Return True
        Else
            Return False
        End If
    End Function

```

```
End Function
```

```
Private Function TemplateFullName _  
    (ByVal Connection As System.Data.IDbConnection, _  
    ByVal ReportName As String, ByVal Current As String, _  
    ByVal Parameters As String) As String _  
    Implements SpreadServer.Interfaces.ITemplateNamer.FullName
```

```
    Return "C:\MyCustomTemplates\MyCustomTemplate.xls"
```

```
End Function
```

```
Private Function FileShortName _  
    (ByVal Connection As System.Data.IDbConnection, _  
    ByVal ReportName As String, ByVal Current As String, _  
    ByVal Parameters As String) As String _  
    Implements SpreadServer.Interfaces.IFileNamer.ShortName
```

```
    Return "MyCustomFileName.xls"
```

```
End Function
```

```
Private Function Query _  
    (ByVal Connection As System.Data.IDbConnection, _  
    ByVal ReportName As String, ByVal SheetName As String, _  
    ByVal Current As String, _  
    ByVal Parameters As String) As String _  
    Implements SpreadServer.Interfaces.IQuery.Query
```

```
    Return "SELECT * FROM SOME_TABLE"
```

```
End Function
```

```
Private Sub Generate _  
    (ByVal Connection As System.Data.IDbConnection, _  
    ByVal ReportName As String, ByVal Workbook As Excel.Workbook, _  
    ByRef Abort As Boolean, ByVal Parameters As String) _  
    Implements SpreadServer.Interfaces.IGenerator.Generate
```

```
    Dim Worksheet As Excel.Worksheet
```

```
    Worksheet = Workbook.Worksheets(1)
```

```
    Worksheet.Cells(1, 1) = "My custom spreadsheet population."
```

```
End Sub
```

```
Private Function Destination _  
    (ByVal Connection As System.Data.IDbConnection, _  
    ByVal ReportName As String, _  
    ByVal Current As String, _  
    ByVal Parameters As String) As String _  
    Implements SpreadServer.Interfaces.IRouter.Destination
```

```
    Return "C:\MyCustomDestinationFolder"
```

```

End Function

Private Sub Compose _
    (ByVal Connection As System.Data.IDbConnection, _
    ByVal ReportName As String, _
    ByVal Message As System.Web.Mail.MailMessage, _
    ByVal Parameters As String) _
    Implements SpreadServer.Interfaces.IMailer.Compose

    'NOTE: All unassigned Message properties default to the
    'values as configured in Spread Server.

    Message.Subject = "My custom email subject"

End Sub

End Class

```

For the purposes of this example, the “SpreadServer.Interfaces.IGenerator.Generate” interface method will be implemented in both C# and VB.NET such that the resulting spreadsheet will be identical, in terms of data, to the spreadsheet that was generated via a simple SQL statement in the previous section. Below are the implementations of the “IGenerator.Generate” interface methods.

C# SpreadServer.Interfaces.IGenerator.Generate Implementation:

```

using System;
using Excel = Microsoft.Office.Interop.Excel;

namespace SpreadServer.SnapIns.ExampleCS
{
    public class SnapIn : SpreadServer.Interfaces.IGenerator
    {
        public SnapIn()
        {
        }

        private void PopulateDivision
            (System.Data.IDbConnection connection, Excel.Worksheet sheet,
            int row, string division)
        {
            System.Data.IDbCommand command;
            System.Data.IDataReader reader;
            string query;

            for (int column=2; column<=13; column++)
            {
                query = "";
                query += "SELECT Revenue FROM DivisionRevenue ";
                query += "WHERE [Division] = \' " + division + "\' ";
            }
        }
    }
}

```

```
query += "AND [Year] = " + System.DateTime.Now.Year.ToString() + " ";
query += "AND [Month] = " + (column - 1);
```

```
command = connection.CreateCommand();
command.CommandText = query;
reader = command.ExecuteReader();
```

```
if (reader.Read())
{
    sheet.Cells[row, column] = reader["Revenue"];
}
```

```
reader.Close();
```

```
}
```

```
/// <summary>
/// IGenerator.Generate method is one of the several
/// Spread Server extensibility interface methods that you
/// can implement. This particular interface method allows
/// for custom spreadsheet population and formatting.
/// </summary>
/// <remarks>
/// If an unhandled run-time error occurs in your code it will
/// bubble up and be handled by Spread Server and Spread Server
/// will shut down gracefully.
/// </remarks>
```

```
void SpreadServer.Interfaces.IGenerator.Generate
(System.Data.IDbConnection Connection,
 string ReportName, Excel.Workbook Workbook,
 ref bool Abort, string Parameters)
{
    Excel.Worksheet sheet;
    Excel.Range cell;
    int lastRow;

    sheet = (Excel.Worksheet)Workbook.Sheets[1];

    this.PopulateDivision(Connection, sheet, 2, "Eastern");
    this.PopulateDivision(Connection, sheet, 3, "Western");

    sheet.get_Range(sheet.Cells[1, 1], sheet.Cells[6, 14]).Columns.AutoFit();

    lastRow = sheet.Cells.SpecialCells(Excel.XlCellType.xlCellTypeLastCell,
        Type.Missing).Row;

    cell = (Excel.Range)sheet.Cells[lastRow + 2, 3];

    cell.Value2 = "Last auto-update: " + DateTime.Now.ToShortDateString() + " "
        + DateTime.Now.ToShortTimeString();

    cell.Font.ColorIndex = 10;
}
}
```

VB.NET SpreadServer.Interfaces.IGenerator.Generate Implementation

```
Imports Excel = Microsoft.Office.Interop.Excel

Public Class SnapIn
    Implements SpreadServer.Interfaces.IGenerator

    Private Sub PopulateDivision _
        (ByVal Connection As System.Data.IDbConnection, _
         ByVal Sheet As Excel.Worksheet, _
         ByVal Row As Integer, ByVal Division As String)

        Dim Command As System.Data.IDbCommand
        Dim Reader As System.Data.IDataReader

        Dim Column As Integer
        Dim Query As String

        For Column = 2 To 13

            Query = ""
            Query &= "SELECT Revenue FROM DivisionRevenue "
            Query &= "WHERE [Division] = '" & Division & "' "
            Query &= "AND [Year] = " & System.DateTime.Now.Year.ToString & " "
            Query &= "AND [Month] = " & (Column - 1).ToString

            Command = Connection.CreateCommand
            Command.CommandText = Query
            Reader = Command.ExecuteReader

            If Reader.Read Then Sheet.Cells(Row, Column) = Reader("Revenue")

            Reader.Close()

        Next Column

    End Sub

    ''' <summary>
    ''' IGenerator.Generate method is one of the several
    ''' Spread Server extensibility interface methods that you
    ''' can implement. This particular interface method allows
    ''' for custom spreadsheet population and formatting.
    ''' </summary>
    ''' <remarks>
    ''' If an unhandled run-time error occurs in your code it will
    ''' bubble up and be handled by Spread Server and Spread Server
    ''' will shut down gracefully.
    ''' </remarks>

    Public Sub Generate _
        (ByVal Connection As System.Data.IDbConnection, _
         ByVal ReportName As String, _
```

```

ByVal Workbook As Excel.Workbook, _
ByRef Abort As Boolean, _
ByVal Parameters As String) _
Implements SpreadServer.Interfaces.IGenerator.Generate

Dim Sheet As Excel.Worksheet
Dim Cell As Excel.Range
Dim LastRow As Integer

Sheet = Workbook.Sheets(1)

Me.PopulateDivision(Connection, Sheet, 2, "Eastern")
Me.PopulateDivision(Connection, Sheet, 3, "Western")

Sheet.Range(Sheet.Cells(1, 1), Sheet.Cells(6, 14)).Columns.AutoFit()

LastRow = Sheet.Cells.SpecialCells(Excel.XlCellType.xlCellTypeLastCell).Row

Cell = Sheet.Cells(LastRow + 2, 3)

Cell.Value = "Last auto-update: " & Now.ToShortDateString & " " _
            & Now.ToShortTimeString

Cell.Font.ColorIndex = 10

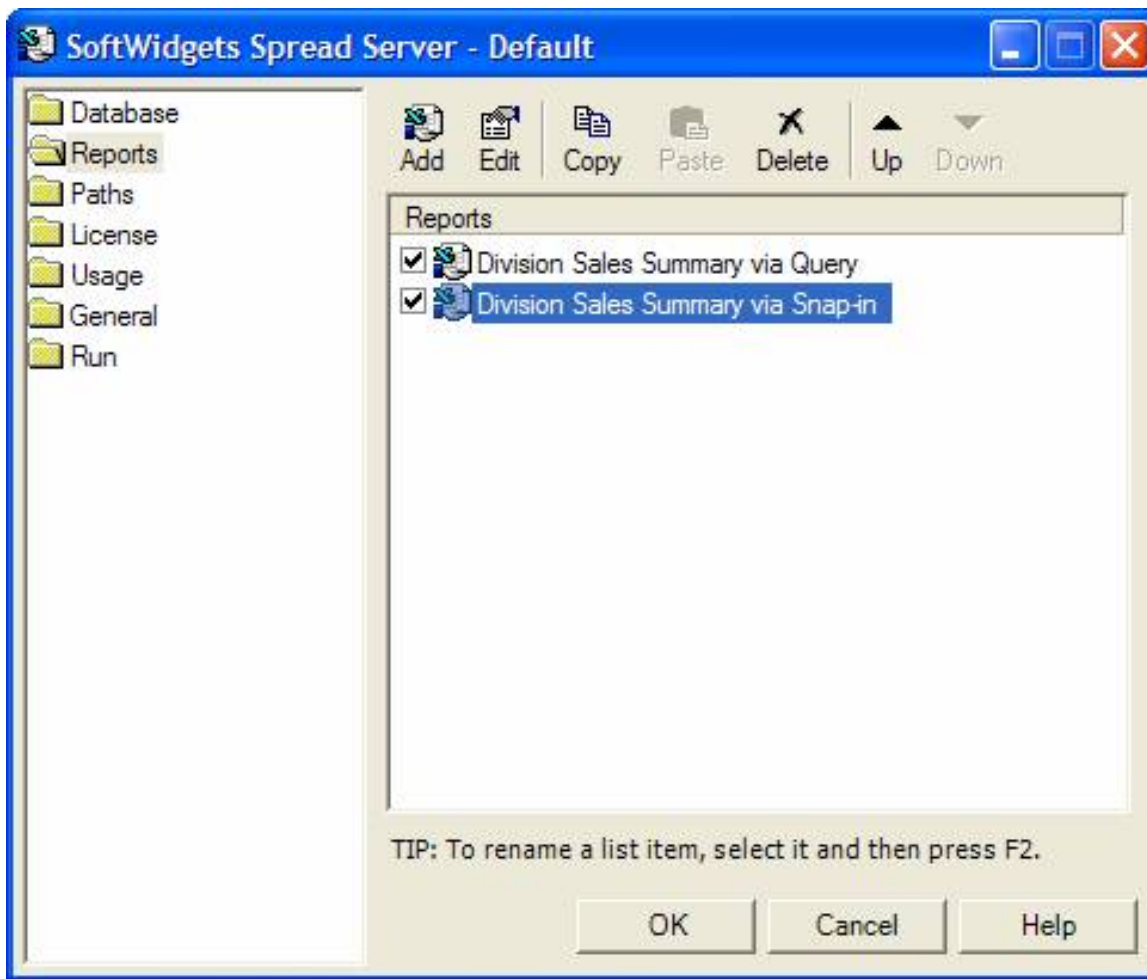
End Sub

```

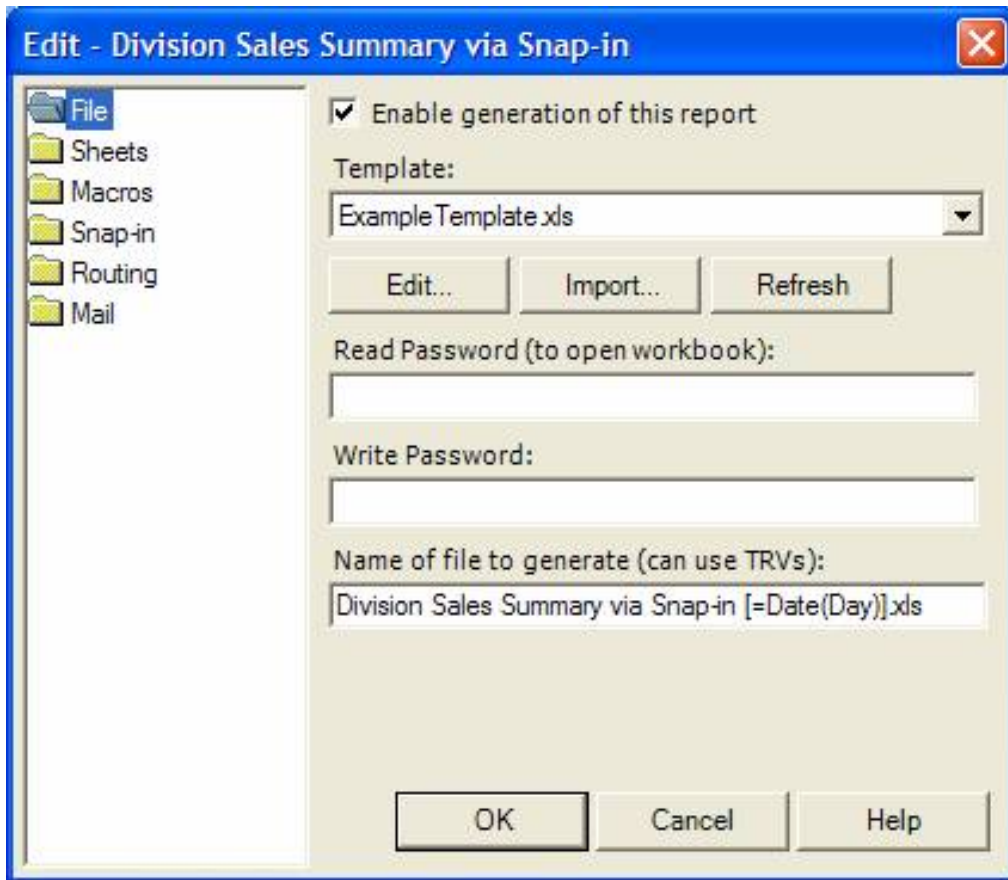
Implementing a snap-in solution requires that you write a snap-in, similar to the examples above to handle desired aspects of the spreadsheet generation process. Snap-ins can be created as a DLL in Visual Studio. Included in the zip file that accompanies this document are two Visual Studio DLL projects, in both C# and VB.NET, that contain the example code show above. The DLLs contained in the zip file will be used to demonstrate how to configure Spread Server snap-ins.

For brevity, some Spread Server configuration steps have been omitted with the assumption that you have read through the previous section of this document. The steps to configure Spread Server to use a snap-in are as follows:

1. Copy the example snap-in DLLs and supporting files to a location on your hard drive. Contained in the zip file that accompanies this document is a folder called "ExampleSnapIns". In the "ExampleSnapIns" folder there is a subfolder called "Compiled" that contains the compiled snap-ins and all the necessary supporting DLLs. For the purposes of this example copy these files to a folder on your hard drive named "C:\Test\SpreadServer\SnapIns".
2. Add another report named "Division Summary via Snap-In" as seen below.



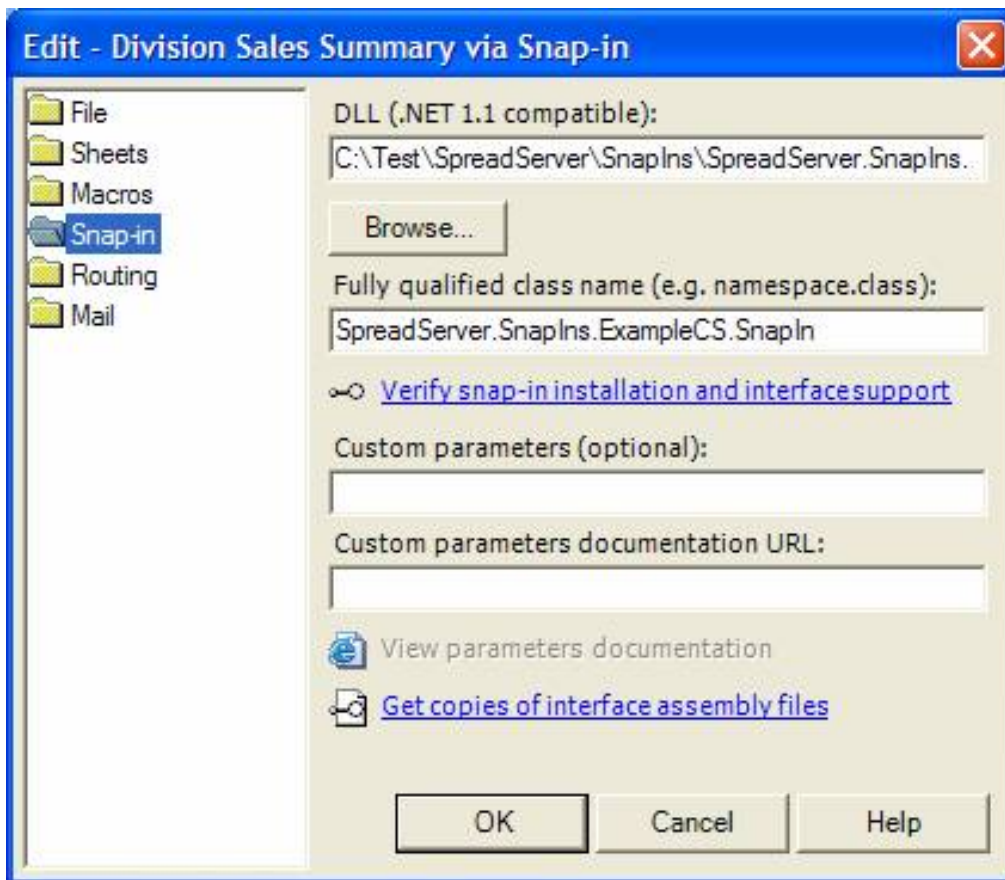
3. Click on the “Edit” button and the report editor will appear. Configure as seen below.



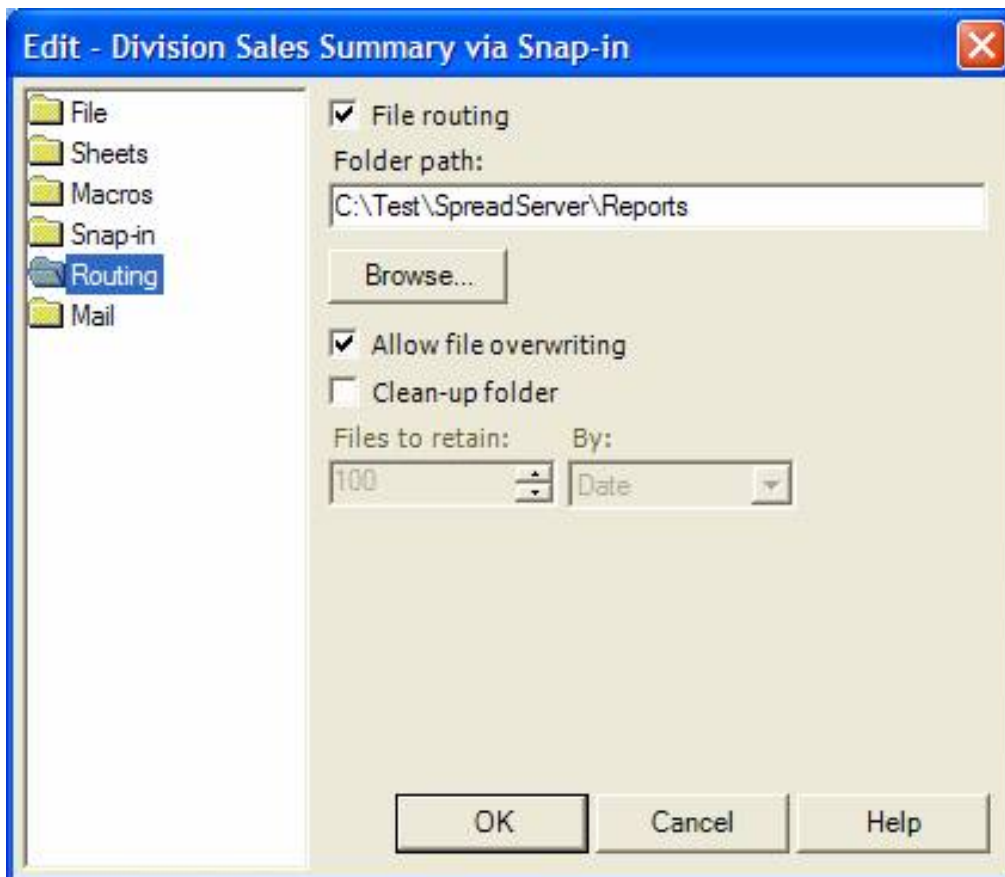
4. Click on the “Snap-in” properties folder. The “Snap-in” property editor allows you to specify the DLL file and the class that implements one or more of the Spread Server interfaces. Configure the properties as seen below.

NOTE: The full path shown below in the DLL field is
“C:\Test\SpreadServer\SnapIns\SpreadServer.SnapIns.ExampleCS.dll”

NOTE FOR VB.NET DEVELOPERS: If you are a VB.NET developer you can enter
“C:\Test\SpreadServer\SnapIns\SpreadServer.SnapIns.ExampleVB.dll” in the DLL field and “SpreadServer.SnapIns.ExampleVB.SnapIn” in the class name field.



5. Click on the "Routing" properties folder and configure as seen below.



6. Click OK. The main Spread Server editing screen will receive the focus.

This completes the Spread Server configuration steps and we are now ready to run the process to generate the spreadsheet.

7. Pick up from step 17 of the simple SQL statement implementation discussed in the previous section and manually create the reports.

Conclusion

In conclusion, Spread Server is capable of handling just about any spreadsheet generation requirements a business may have. From the simple SQL statement based implementations to the more sophisticated snap-in based implementations, reporting projects can now be completed in hours, not months or years as with competing products. If you have not already done so, SoftWidgets invites you to try our unique products. See the power for yourself by [downloading a free evaluation copy of Spread Server](#).