Drill-Through System Documentation

Hyperion Planning Drill-Through using Essbase Drill-Through Definitions and ASP.NET

**Document Prepared By**

|  |  |
| --- | --- |
| **Document Owners** | **Role** |
| Ray Lin | ASP.NET Development |
| Ahmed Shaffi | Essbase Development |
| **Contributing Content Owners** | **Role** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Table of Contents

[1 Default.aspx 3](#_Toc392086972)

[1.1 Overview 3](#_Toc392086973)

[1.1.1 Summary 3](#_Toc392086974)

[1.1.2 System Flow 3](#_Toc392086975)

[1.2 Views 3](#_Toc392086976)

[1.2.1 Default View 3](#_Toc392086977)

[1.3 Functions 5](#_Toc392086978)

[1.3.1 getBu(String entity) 5](#_Toc392086979)

[1.3.2 parsePost(string postString) 6](#_Toc392086980)

[1.3.3 exportExcel(HashTable parsedHash, List<GridView> databaseTables) 7](#_Toc392086981)

[1.3.4 excelButton\_Click(object Sender, EventArgs e) 9](#_Toc392086982)

[1.3.5 In-line Procedural Code on Default.aspx 10](#_Toc392086983)

[2 Data.aspx 10](#_Toc392086984)

[2.1 Overview 10](#_Toc392086985)

[2.1.1 Summary 10](#_Toc392086986)

[2.1.2 System Flow 11](#_Toc392086987)

[2.2 Views 11](#_Toc392086988)

[2.3 Functions 11](#_Toc392086989)

[2.3.1 Page\_Load(object sender, EventArgs e) 11](#_Toc392086990)

[3 MakeTable.aspx 13](#_Toc392086991)

[3.1 Overview 13](#_Toc392086992)

[3.1.1 Summary 13](#_Toc392086993)

[3.1.2 System Flow 14](#_Toc392086994)

[3.2 Views 14](#_Toc392086995)

[3.2.1 Default View 14](#_Toc392086996)

[3.3 Functions 15](#_Toc392086997)

[3.3.1 rLoadTopNode(String filename) 15](#_Toc392086998)

[3.3.2 isLevelZero(XmlNode currentNode) 15](#_Toc392086999)

[3.3.3 stopAndHash(XmlNode currentNode) 16](#_Toc392087000)

[3.3.4 checkForBU(string CCstring) 17](#_Toc392087001)

[3.3.5 areChildrenNonMembers(XmlNode currentNode) 17](#_Toc392087002)

[3.3.6 rNodeTest(XmlNode node, List<Tuple<string, string>> rTableTup) 18](#_Toc392087003)

[3.3.7 writeToDrillDB(List<Tuple<string, string>> listOfPairs) 18](#_Toc392087004)

[3.3.8 Button1\_Click(object sender, EventArgs e) 20](#_Toc392087005)

# Default.aspx

## Overview

### Summary

This application takes a HTTP post value sent from Hyperion Essbase and parses that information to create a query that retrieves the relevant Essbase intersection from the source database.

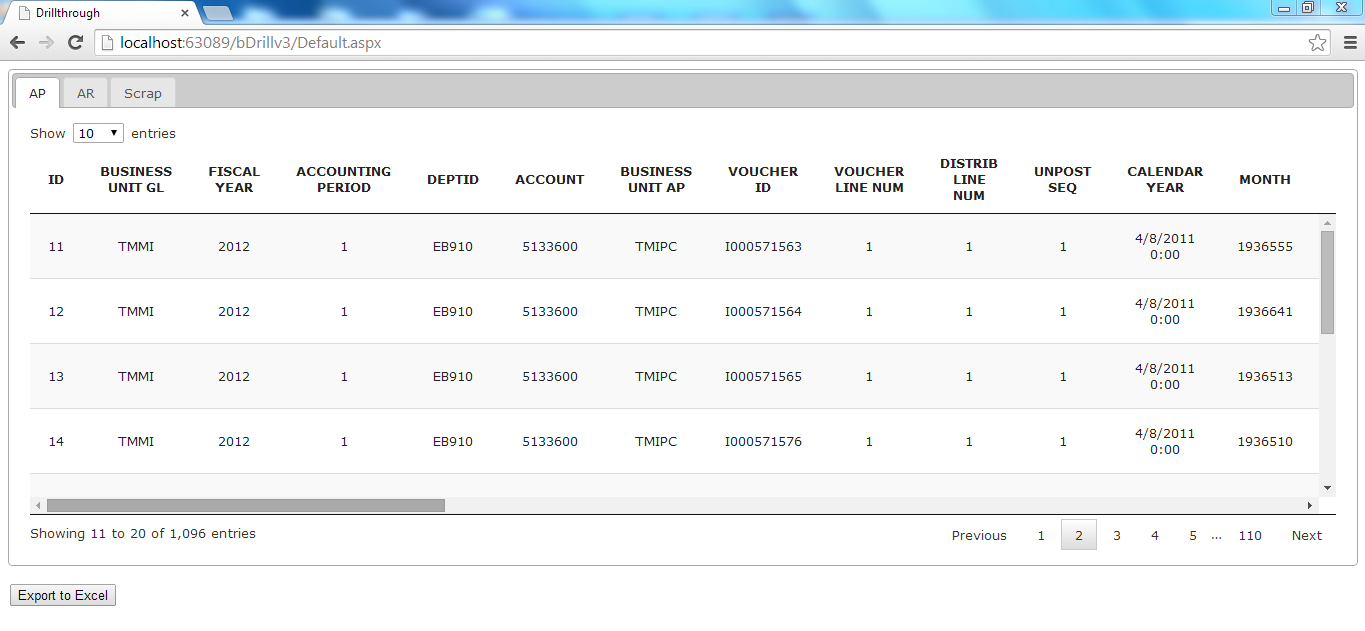
### System Flow



## Views

### Default View

The default view displays data retrieved from the database in a HTML table element that is styled and modified by a jQuery DataTables element. These tables are stored in a jQuery UI tab view set.



#### Page Elements

##### HTML Table Elements

HTML table elements are the core of the display elements of this page. The HTML table that is used on this page is a standard HTML table that is then modified by the DataTables jQuery plugin. Each database table that will be queried should have its own HTML table on the page and each should have a unique id element. This id element will be used by the DataTables plugin to identify which HTML table to modify. However, all tables will need to be of the CSS class “display” to use the correct DataTables styling rules.

Note that the HTML tables on the page only include the <thead> elements and that the <th> elements are programmatically generated by a call to the generateTableHeaders() function. The reason that there are no <tbody> elements is that the table body will be generated by DataTables through an AJAX call to the data source so no <tbody> or <td> elements are needed in the HTML table.

###### Source

<table class="display" id="checkbook" cellspacing="0" width="100%">

                <thead>

                    <tr>

                        <%= generateTableHeaders("checkbook") %>

                    </tr>

                </thead>

            </table>

##### jQuery DataTables Plugin

This jQuery plugin modifies the HTML table elements to provide the ability to display and paginate records. DataTables will need to be called on every HTML table on the page. DataTables will also need to have the following parameters:

"scrollX": true, //sets horizontal scroll

       "ordering": false, //turn off ordering by clicking on headers

       "scrollY": "45vh", //vertical scroll set to 45% of view height

       "processing": true,

     "serverSide": true, //turn on server side processing

     "filter": false, //turn off search filter. can be implemented with SQL changes

     "ajax": {

         "url": "../rDrillv3/Data.aspx", //url to webservice

        "type": "POST", //using post instead of default of GET bc asp.net cant handle large query strings

        "data": function (d) { //send in all dimension values as parameters

              d.BUSINESS\_UNIT\_GL = document.getElementById('BUSINESS\_UNIT\_GL').value;

              d.Entity = document.getElementById('Entity').value;

              d.Account = document.getElementById('Account').value;

              d.FISCAL\_YEAR = document.getElementById('FISCAL\_YEAR').value;

              d.tableName = "checkbook";

###### Source

$(document).ready(function () {

            $('#checkbook').DataTable({

                "scrollX": true, //sets horizontal scroll

                "ordering": false, //turn off ordering my clicking on headers. can be implemented with SQL alterations

                "scrollY": "45vh", //veritcal scroll set to 45% of view height

                "processing": true,

                "serverSide": true, //turn on server side processing

                "filter": false, //turn off search filter. can be implemented with SQL changes

                "ajax": {

                    "url": "../rDrillv3/Data.aspx", //url to webservice to return server side processed data

                    "type": "POST", //using post instead of default of GET bc asp.net cant handle large query strings

                    "data": function (d) { //send in all dimension values as parameters

                        d.BUSINESS\_UNIT\_GL = document.getElementById('BUSINESS\_UNIT\_GL').value;

                        d.Entity = document.getElementById('Entity').value;

                        d.Account = document.getElementById('Account').value;

                        d.FISCAL\_YEAR = document.getElementById('FISCAL\_YEAR').value;

                        d.tableName = "checkbook";

                    }

                }

            }); //end datatables

        });      //end ready

##### jQuery UI Tabs

This set of jQuery UI tabs are used to display multiple HTML tables on one page to be able to display multiple datasets in one retrieve.

###### Source

**jQuery Initialization:**

$(function () {

$("#tabs").tabs();

});

**HTML:**

<div id="tabs" style="font-size: small; font-family: Verdana; text-align: center">

        <ul>

            <li><a href="#tabs-1">AP</a></li>

            <li><a href="#tabs-2">AR</a></li>

       </ul>

        <div id="tabs-1">

            <table class="display" id="checkbook" cellspacing="0" width="100%">

                <thead>

                    <tr>

                        <%= generateTableHeaders("checkbook") %>

                    </tr>

                </thead>

            </table>

        </div>

        <div id="tabs-2">

            <table class="display" id="checkbook2" cellspacing="0" width="100%">

                <thead>

                    <tr>

                        <%= generateTableHeaders("checkbook2") %>

                    </tr>

                </thead>

            </table>

        </div>

## Functions

### getBu(String entity)

#### Overview

This function is designed to check the “Entity.id” value from the Essbase post data to see if it contains entity information. In P26/P28, the level zero entity members are designed to have a 2 letter representation of the NAMC entity followed by “\_CC”. For example, the cost center EB000 for TMMTX will be “EB000\_TX\_CC”. When passed a string, this function will check if it contains entity information and will return the unabbreviated entity value.

#### Parameters

*entity*

The string to be checked. The function was designed to check the level zero member of the entity dimension.

#### Return Value

Type: String

The unabbreviated entity name. Ex. “TMMI” if passed either “EB001\_MI\_CC” or “TMMI”

#### Source Code

protected string getBU(string entity)

    {

        if (entity.Contains("\_MI") || entity.Contains("TMMI"))

        {

            return "TMMI";

        }

        else if (entity.Contains("\_AL") || entity.Contains("TMMAL"))

        {

            return "TMMAL";

        }

        else if (entity.Contains("\_TX") || entity.Contains("TMMTX"))

        {

            return "TMMTX";

        }

        else if (entity.Contains("\_MS") || entity.Contains("TMMMS"))

        {

            return "TMMMS";

        }

        else if (entity.Contains("\_NK") || entity.Contains("TMMNK"))

        {

            return "TMMNK";

        }

        else if (entity.Contains("\_NJ") || entity.Contains("\_NS") || entity.Contains("\_NT") || entity.Contains("BODINE"))

        {

            return "BODINE";

        }

        //need to add the abbrev for TABC

        else if (entity.Contains("TABC"))

        {

            return "TABC";

        }

        else if (entity == "Total\_Plant\_Rollup")

        {

            return "TMMAL\' OR \'TMMI\' OR \'TMMMS\' OR \'TMMWV\' OR \'TMMTX\' OR \'TMMK\' OR \'TMMNK\' OR \'BODINE\' OR \'TABC";

        }

        else

        {

            return entity;

        }

    }

### parsePost(string postString)

#### Overview

The intent of this function is to take the POST data passed from Essbase (passed as a single string) and split up the key/value pairs within that string into a hashtable object. When Essbase sends the POST data, all values are paired with the tag “attribute” and the actual attribute key (ex. Entity, Account, Period, etc) will be in the value associated with “attribute” followed by “.id.” and then followed by the actual value. For example, a sample POST request sent by Essbase with an Entity of “EK100\_MI\_CC”, Version of “working” and Account of “5133600” will be sent as

“attribute=’Entity.id.EK100\_MI\_CC,attribute=’Version.id.working’,attribute=’Account.id.5133600’”. This function will take the string and parse out the key/value pairs and set the pairs as a hashtable. For example, the hashtable created with the POST data above would be {Entity.id : EK100\_MI\_CC, Version.id : working, Account : 5133600}. In addition to simply parsing out the key value pairs, this function also performs some transformations on the values where needed. For example, it will call the getBU() function to convert the Hyperion entity names to the Checkbook entity names and it will also convert the year format from Hyperion to match that of Checkbook.

#### Parameters

*postString*

This parameter is a string that contains the POST data values of items belonging to attribute”. An example string would be: “Entity.id.EK100\_MI\_CC,Version.id.working,Account.5133600”

#### Return Value

*Type*: Hashtable

This function returns a hashtable with the values of inputted string as key/value pairs. An example would be {Entity.id : EK100\_MI\_CC, Version.id : working, Account : 5133600}.

#### Source Code

protected Hashtable parsePost(string postString)

    {

        if (postString == "" || postString == null)

        {

            return null;

        }

        else

        {

            string[] splitString = postString.Split(new Char[] { ',' });

            Hashtable postHash = new Hashtable();

            for (int i = 0; i <= splitString.Length - 1; i++)

            {

postHash.Add(splitString[i].Substring(0, splitString[i].LastIndexOf(".")), splitString[i].Substring(splitString[i].LastIndexOf(".") + 1));

            }

            postHash.Add("BUSINESS\_UNIT\_GL.id", getBU(postHash["Entity.id"].ToString()));

            postHash["Years.id"] = postHash["Years.id"].ToString().Replace("FY", "20");

            return postHash;

        }

    }

### exportExcel(HashTable parsedHash, List<string> databaseTables)

#### Overview

This function receives the POST data from Essbase and a List object of all the database names in order to re-query the database. The results of the query are stored in DataTable objects that are then stored in a DataSet object. Using the EPPlus library (namespace OfficeOpenXml), the function converts the DataSet into an Excel workbook with each DataTable within the workbook as a new sheet. The function then returns the data to the page as an Excel file.

#### Parameters

*parsedHash*

This parameter is a HashTable object that contains the intersection values and the dimension names stored in key value pairs.

*databaseTables*

This is a List<string> object that stores all the database table names as strings

#### Return Value

None.

#### Source Code

using OfficeOpenXml;

protected void exportExcel(Hashtable parsedHash, List<string> databaseTables)

    {

        if (parsedHash == null)

        {

            return;

        }

        else

        {

            string filelocation = Server.MapPath("fakeCheckbook.accdb");

            string ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + filelocation;

            DataSet ds = new DataSet();

            foreach (string databaseTable in databaseTables)

            {

                string query = "SELECT \* FROM " + databaseTable + " WHERE (BUSINESS\_UNIT\_GL = '" + parsedHash["BUSINESS\_UNIT\_GL.id"] + "') AND DEPTID IN (SELECT DISTINCT levelZero FROM rDrill WHERE parent = '" + parsedHash["Entity.id"] + "') AND ACCOUNT IN (SELECT DISTINCT levelZero FROM rDrill WHERE parent = '" + parsedHash["Account.id"] + "') AND FISCAL\_YEAR='" + parsedHash["Years.id"] + "';";

                OleDbConnection connection = new OleDbConnection(ConnectionString);

                OleDbCommand command = new OleDbCommand(query, connection);

                connection.Open();

                OleDbDataAdapter adapter = new OleDbDataAdapter(command);

                DataTable dt = new DataTable();

                adapter.Fill(dt);

                ds.Tables.Add(dt);

            }

            ExcelPackage package = new ExcelPackage();

            foreach (DataTable table in ds.Tables)

            {

                ExcelWorksheet worksheet = package.Workbook.Worksheets.Add(table.TableName);

                for (int i = 1; i < table.Columns.Count + 1; i++)

                {

                    worksheet.Cells[1, i].Value = table.Columns[i - 1].ColumnName;

                }

                for (int j = 0; j < table.Rows.Count; j++)

                {

                    for (int k = 0; k < table.Columns.Count; k++)

                    {

                        worksheet.Cells[j + 2, k + 1].Value = table.Rows[j].ItemArray[k].ToString();

                    }

                }

            }

            //Read the Excel file in a byte array. here pck is the Excelworkbook

            Byte[] fileBytes = package.GetAsByteArray();

            //Clear the response

            Response.Clear();

            Response.ClearContent();

            Response.ClearHeaders();

            Response.Cookies.Clear();

            //Add the header & other information

            Response.Cache.SetCacheability(HttpCacheability.Private);

            Response.CacheControl = "private";

            Response.Charset = System.Text.UTF8Encoding.UTF8.WebName;

            Response.ContentEncoding = System.Text.UTF8Encoding.UTF8;

            Response.AppendHeader("Content-Length", fileBytes.Length.ToString());

            Response.AppendHeader("Pragma", "cache");

            Response.AppendHeader("Expires", "60");

            Response.AppendHeader("Content-Disposition",

            "attachment; " +

            "filename=\"drill-" + DateTime.Now.ToString("yyyyMMddHHmmss") + ".xlsx\"; " +

            "size=" + fileBytes.Length.ToString() + "; " +

            "creation-date=" + DateTime.Now.ToString("yyyyMMddHHmmss") + "; " +

            "modification-date=" + DateTime.Now.ToString("yyyyMMddHHmmss") + "; " +

            "read-date=" + DateTime.Now.ToString("yyyyMMddHHmmss"));

            Response.ContentType = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";

            //Write it back to the client

            Response.BinaryWrite(fileBytes);

            Response.End();

        }

    }

### excelButton\_Click(object Sender, EventArgs e)

#### Overview

This function runs when the “Export to Excel” button is clicked. The function first builds the parameters necessary for to call the exportExcel() function. The objects necessary for the parameters are the HashTable parsedHash that contains the dimension name and dimension value stored in key value pairs and also the List<string> object that contains a list of all the database table names that need exporting to Excel. The function firsts creates a string called postString that is read from a HiddenField object on the page called hiddenPost which contains the Essbase POST request data stored as a single string. This string is then parsed to create the parsedHash HashTable object. This object along with the database table names are then used to call the exportExcel function.

#### Parameters

There are no parameters other than the standard “object Sender” and “EventArgs e” parameters which are required by .NET to handle events. Note however, that the HiddenField object hiddenPost is used like a parameter in that it is used to pass a value from the page to this function. This workaround was used because it was not possible to pass the necessary objects from the ASP button to the server code.

#### Return Value

None.

#### Source Code

protected void excelButton\_Click(object sender, EventArgs e)

    {

        string postString = hiddenPost.Value;

        Hashtable parsedHash = parsePost(postString);

        List<string> databaseTables = new List<string>();

        databaseTables.Add("checkbook");

        databaseTables.Add("checkbook2");

        exportExcel(parsedHash, databaseTables);

    }

### generateTableHeaders(string tableName)

#### Overview

This function queries the source database tables for the table column names. These column names are then used to generate the HTML table header information in the that would be in the <th> tags. Note that this function only create the information in <th> tags. The <table> and <tr> tags will not be generated by this code. The actual table body and data will be generated by the jQuery DataTables element.

#### Parameters

*tableName*

The tableName parameter is a string that contains the name of the table to be queried.

#### Return Value

Type: String

The return value is a string that contains all of the table column field names that are each enclosed in <th> </th> tags.

#### Source Code

protected string generateTableHeaders(string tableName)

    {

        string filelocation = Server.MapPath("fakeCheckbook.accdb");

        string ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + filelocation;

        //this is just to select column names and is not ideal sql but is the only option for Access

        //this should be changed when using a real DB

        string query = "SELECT \* FROM " + tableName + " WHERE 1=2;";

        OleDbConnection connection = new OleDbConnection(ConnectionString);

        OleDbCommand command = new OleDbCommand(query, connection);

        connection.Open();

        OleDbDataAdapter adapter = new OleDbDataAdapter(command);

        DataTable dt = new DataTable();

        adapter.Fill(dt);

        //to remove the underscores in the column names, if desired

        foreach (DataColumn column in dt.Columns)

        {

            column.ColumnName = column.ColumnName.Replace("\_", " ");

        }

        //to generate the <th> string

        string tableHeaderString = "";

        foreach (DataColumn column in dt.Columns)

        {

            tableHeaderString = tableHeaderString + " <th> " + column.ColumnName + " </th>";

        }

        return tableHeaderString;

    }

### In-line Procedural Code on Default.aspx

#### Overview

Default.aspx contains in-line procedural code that collects the POST data sent from Essbase and parses that data using the parsePost() function and sets what is returned by parsePost into a hashtable. This hashtable is then broken up by its key/value pairs and set into a series of HiddenField objects on the page. These HiddenField objects will then be accessed by the DataTables jQuery plugin as part of its AJAX requests to the Data.aspx web service for data retrieval.

Additionally, the code checks if there is no POST data to print an error message to the page and also sets the HiddenField value hiddenPost as the POST data from Essbase. This HiddenField is used to call the exportExcel() function when the Export to Excel button is clicked.

#### Source Code

<head>

<%

        string postString = Page.Request.Form["attribute"];

        Hashtable parsedHash = parsePost(postString);

        //set the hidden fields to send the parsed values to the web service through DataTables

        BUSINESS\_UNIT\_GL.Value = parsedHash["BUSINESS\_UNIT\_GL.id"].ToString();

        Entity.Value = parsedHash["Entity.id"].ToString();

        Account.Value = parsedHash["Account.id"].ToString();

        FISCAL\_YEAR.Value = parsedHash["Years.id"].ToString();

    %>

</head>

<body>

    <%

        if (Page.Request.Form["attribute"] == null)

        {

            Response.Write("<p>NO POST DATA RECEIVED FROM ESSBASE</p>");

        }

        hiddenPost.Value = Page.Request.Form["attribute"];

    %>

</body>

# Data.aspx

## Overview

### Summary

Data.aspx is a web service made specifically to interact with the DataTables jQuery plugin API. This service receives POST data generated by the DataTables element on Default.aspx through an AJAX XmlHttpRequest. This request includes the Essbase intersection information along with the table pagination information. This information is then used to query the database for the data rows that are specific to the paginated draw on the current page of the DataTables element. The queried data is then encoded in JSON and sent back to the DataTables element on Default.aspx.

### System Flow



## Views

None.

## Functions

### Page\_Load(object sender, EventArgs e)

#### Overview

The Page\_Load function is the standard ASP.NET function that runs on every page load. In this case, the function is used to read POST data sent from a DataTables element. Note that by default, DataTables sends data to a web service using a GET request but this service is configured to use a POST request because of limitations with ASP.NET. Because of this change, any DataTable element that calls this web service will need to be configured to send a POST request instead of the default GET request.

This service will collect the length in number of records that DataTables expects, the start point of the record set that will be returned and the Essbase intersection information. Note that the draw parameter is a required element of DataTables that needs to be collected and returned back to DataTables.

After the DataTable POST parameters are collected, the parameters are then used to query the database for the correct rows. Note that in its current implementation, the code queries all of the data for the specified intersection and then takes only the rows required of the page to send back to DataTables. A more efficient method would be to only query the database for the rows needed. This was not implemented in this current state because of the limitations with MS Access, the current test database. When the system can be used with a more appropriate database, the routine should be changed to only query what is needed through the proper SQL retrieve.

After the data is retrieved, it is set to a C# DataTable object and then the appropriate rows are extracted from the total retrieve and this is set as an object array of object arrays. This array of arrays is then set with other parameters needed by DataTables into the “result” object. This object includes:

1. recordsTotal: the number of total records of this retrieve (not just this page)
2. recordFiltered: the number of records after filtering (if performed)
3. draw: the draw parameter that needs to be sent back to DataTables
4. data: the row information as an array of arrays

The “result” object is then encoded into JSON using the JavaScriptSerializer serialize method and retuned back to DataTables through the server response.

#### Parameters

There are no other function parameters other than the standard “object Sender” and “EventArgs e” parameters which are required by .NET to handle events. However, the routine will collect information sent from DataTables through a POST request. These POST parameters are described below:

var length: the number of records of the current draw

        var start: the start point of the current draw

        var draw: a draw counter that is sent by DataTables that will need to be returned to datatables

var BUSINESS\_UNIT\_GL: the BU of the Essbase intersection

        var Entity: the entity of the Essbase intersection

        var Account: the account number of the Essbase intersection

        var FISCAL\_YEAR: the format adjusted fiscal year of the Essbase intersection

        var tableName: the table to be queried

#### Return Value

None.

#### Source Code

public void Page\_Load(object sender, EventArgs e)

    {

        // Required parameters sent from DataTables

        var length = int.Parse(Request.Form["length"]);

        var start = int.Parse(Request.Form["start"]);

        var draw = Request.Form["draw"];

        //custom parameters from DataTalbes

        var BUSINESS\_UNIT\_GL = Request.Form["BUSINESS\_UNIT\_GL"];

        var Entity = Request.Form["Entity"];

        var Account = Request.Form["Account"];

        var FISCAL\_YEAR = Request.Form["FISCAL\_YEAR"];

        var tableName = Request.Form["tableName"];

        // query all data. when doing for real should probabaly only query the relevant range. right now cant do in sql

        // because of limitation with access

        string filelocation = System.Web.HttpContext.Current.Server.MapPath("fakeCheckbook.accdb");

        string ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + filelocation;

        string query = "SELECT \* FROM " + tableName + " WHERE (BUSINESS\_UNIT\_GL = '" + BUSINESS\_UNIT\_GL + "') AND DEPTID IN (SELECT DISTINCT levelZero FROM rDrill WHERE parent = '" + Entity + "') AND ACCOUNT IN (SELECT DISTINCT levelZero FROM rDrill WHERE parent = '" + Account + "') AND FISCAL\_YEAR='" + FISCAL\_YEAR + "';";

        OleDbConnection connection = new OleDbConnection(ConnectionString);

        OleDbCommand command = new OleDbCommand(query, connection);

        connection.Open();

        OleDbDataAdapter adapter = new OleDbDataAdapter(command);

        DataTable dt = new DataTable();

        adapter.Fill(dt);

        connection.Close();

        //prep data to be stored in the "result" var

        //the dt data must be but into an array of arrays

        //right now the line below puts data in an array of DataRows which will not be read correctly by the JSON writer

        var tempdata = dt.Select().Skip(start).Take(length).ToArray();

        object[][] drawArray = new object[tempdata.Length][];

        int i = 0;

        foreach (DataRow row in tempdata)

        {

            drawArray[i] = row.ItemArray;

            i++;

        }

        //put the response expected by Datatables into a var "result"

        var result = new

        {

            recordsTotal = dt.Rows.Count,

            recordsFiltered = dt.Rows.Count,

            draw = draw,

            data = drawArray

        };

        //convert "result" array into JSON and write to response

        var serializer = new JavaScriptSerializer();

        var json = serializer.Serialize(result);

        Page.Response.Clear();

        Page.Response.ClearContent();

        Page.Response.ClearHeaders();

        Page.Response.Cookies.Clear();

        Page.Response.ContentType = "application/json";

        Page.Response.Write(json);

    }

# MakeTable.aspx

## Overview

### Summary

The purpose of this application is to create a table of all level zero members paired with each of its parent nodes. This will allow the drill application to drill at a higher level. This application takes an Essbase XML outline and creates a database table that contains each level zero paired with each of its parent and then with itself.

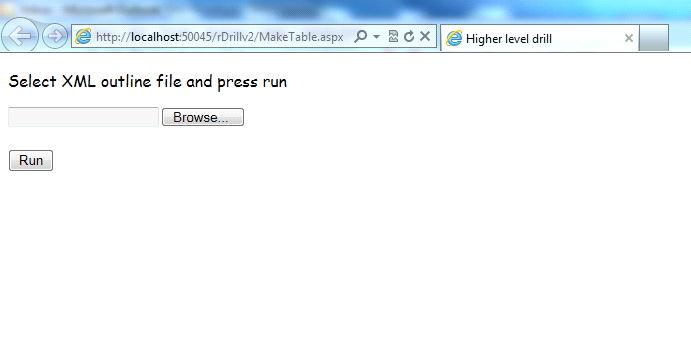
### System Flow



## Views

### Default View

The default view consists of one ASP upload control that allows the user to upload the Essbase XML outline to be processed.



#### Page Elements

##### ASP File Upload Control

This ASP file upload control allows the user to upload the Essbase XML outline to the application server.

###### Source

<asp:FileUpload ID="FileUpload1" runat="server" />

##### ASP Button

This ASP button calls Button1\_Click() that runs the processing of the XML document uploaded to the server.

###### Source

<asp:Button ID="Button1" runat="server" OnClick="Button1\_Click" Text="Run" />

## Functions

### rLoadTopNode(String filename)

#### Overview

This function loads a specified Essbase XML outline on the application server and returns the first node of that XML document.

#### Parameters

*filename*

The file name of the XML file to be loaded.

#### Return Value

Type: XmlNode

The first node of the XML document is returned.

#### Source Code

public XmlNode rloadTopNode(string filename)

{

XmlDocument doc = new XmlDocument();

doc.Load(Server.MapPath(filename));

XmlNode node = doc.DocumentElement;

return node;

}

### isLevelZero(XmlNode currentNode)

#### Overview

This is a Boolean function that tests whether the current node is or is not a level zero member. If the selected node is a level zero member, the function will return true, otherwise it will return false.

#### Parameters

*currentNode*

The XmlNode object to be tested.

#### Return Value

Type: Boolean

*True*: is level zero

*False*: is not a level zero

#### Source Code

public bool isLevelZero(XmlNode currentNode)

{

if (currentNode == null || currentNode.Name != "Member")

{

return false;

}

else if ((currentNode.HasChildNodes == false && currentNode.Name == "Member") || (areChildrenNonMembers(currentNode) && currentNode.Name == "Member"))

{

return true;

}

else

{

return false;

}

}

### stopAndHash(XmlNode currentNode)

#### Overview

This function will create a List object of Tuple<string, string> objects and when passed an XmlNode object, will set the value of the XmlNode as the first string in the Tuple pairs and match the string with every parent node of that node. Note: this function will also call the checkForBU() function to check the XmlNode originally passed to the function to see if that node contains entity and cost center information. It will then use the value returned from checkForBU() as the first string in the Tuple string pairs.

#### Parameters

*currentNode*

An Xml node to set as the first string in the list of tuples that will be created

#### Return Value

Type: List<Tuple<string, string>>

The function will return a list object that contains string Tuple pairs that will contain the original node passed to the function as the first string and parent nodes as the second string

#### Source Code

public List<Tuple<string, string>> stopAndHash(XmlNode currentNode)

{

List<Tuple<string, string>> listOfPairs = new List<Tuple<string, string>>();

XmlNode lvlZeroNode = currentNode;

string accountStr = lvlZeroNode.Attributes["name"].Value;

string checkedString = checkForBU(accountStr);

currentNode = lvlZeroNode;

//conditional has to be 2 parentnodes up because the top parent has the parent of "Document"

while (currentNode.ParentNode.ParentNode != null)

{

Tuple<string, string> pair = new Tuple<string, string>(currentNode.ParentNode.Attributes["name"].Value, checkedString);

listOfPairs.Add(pair);

currentNode = currentNode.ParentNode;

}

listOfPairs.Add(new Tuple<string, string>(accountStr, checkedString));

return listOfPairs;

}

### checkForBU(string CCstring)

#### Overview

Checks if a string contains the substring “\_CC”. If it does, the function will trim the last 6 characters off of the right of the string and return the string.

#### Parameters

*CCstring*

The string that is to be checked if it contains “\_CC”

#### Return Value

Type: String

If the original string contained the substring “\_CC” then the original string will be shortened by 6 characters from the right. Otherwise, the original unaltered string will be returned.

#### Source Code

public string checkForBU(string CCstring)

{

if (CCstring.Contains("\_CC"))

{

string ccSubString = CCstring.Substring(0,CCstring.Length - 6);

return ccSubString;

}

else

{

return CCstring;

}

}

### areChildrenNonMembers(XmlNode currentNode)

#### Overview

This is a Boolean function that when passed an XmlNode, checks to see if all children of that node are not “member” nodes.

#### Parameters

*currentNode*

The XmlNode to be checked

#### Return Value

Type: Boolean

*True*: Does not have any children that are “member” nodes

*False*: Does have child nodes that are “member” nodes

#### Source Code

public bool areChildrenNonMembers(XmlNode currentNode)

{

foreach (XmlNode node in currentNode)

{

if (node.Name == "Member")

{

return false;

}

}

return true;

}

### rNodeTest(XmlNode node, List<Tuple<string, string>> rTableTup)

#### Overview

This is a function that when passed an XmlNode, will check if each its immediate children nodes is a level zero. If a child node is a level zero node, it will add that level zero node to a List of Tuple<string, string> objects along with all of that level zero node’s parent nodes. If that child is not a level zero node, the function will call itself to run again at that child node.

#### Parameters

*node*

The XmlNode object to be checked

*rTableTup*

A List object of Tuple<string, string> objects that will contain the level zero node paired with its parent node. This List object is then passed back into the function when called recursively to keep a running total of all <level zero, parent> Tuple pairs.

#### Return Value

Type: List<Tuple<string, string>>

This returned value will be a list of pairs of strings that will contain the level zero paired with each of its parents

#### Source Code

public List<Tuple<string, string>> rNodeTest(XmlNode node, List<Tuple<string, string>> rTableTup)

{

foreach (XmlNode currentNode in node)

{

if (isLevelZero(currentNode))

{

foreach (Tuple<string, string> pair in stopAndHash(currentNode))

{

rTableTup.Add(pair);

}

}

else

{

rTableTup = rNodeTest(currentNode, rTableTup);

}

}

return rTableTup;

}

### writeToDrillDB(List<Tuple<string, string>> listOfPairs)

#### Overview

This function takes a List object containing Tuple objects containing pairs of strings and writes each of those pairs to a database where the first string in each Tuple is set to the “parent” field in the database the second string in the pair is set to the “levelZero” field in the database.

#### Parameters

*listOfPairs*

A List object of Tuple<string, string> objects that will contain the level zero node paired with its parent node.

#### Return Value

None.

#### Source Code

protected void writeToDrillDB(List<Tuple<string, string>> listOfPairs)

{

string filelocation = Server.MapPath("fakeCheckbook.accdb");

string ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source="+filelocation;

OleDbConnection connection = new OleDbConnection(ConnectionString);

connection.Open();

//delete everything in the table before loading

OleDbCommand deleteCommand = new OleDbCommand("DELETE \* FROM rDrill;", connection);

deleteCommand.ExecuteNonQuery();

foreach (Tuple<string, string> pair in listOfPairs)

{

string query = "INSERT INTO rDrill (parent, levelZero) VALUES (\"" + pair.Item1 + "\", \"" + pair.Item2 + "\");";

OleDbCommand command = new OleDbCommand(query, connection);

command.ExecuteNonQuery();

}

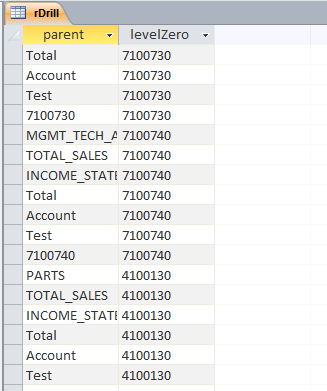
connection.Close();

updateLabel.Text = "Done<br />";

}

#### Other Outputs

This function will write the ordered tuple pairs passed into this function to an external database. Below is an example of the database table that the function will create.



### Button1\_Click(object sender, EventArgs e)

#### Overview

Saves an uploaded file into the application folder.

#### Parameters

*None*

#### Return Value

None

#### Source Code

protected void Button1\_Click(object sender, EventArgs e)

{

if (FileUpload1.HasFile)

{

emptyupload.Text = "";

FileUpload1.SaveAs(Server.MapPath("~/" + FileUpload1.FileName));

updateLabel.Text = "Running....";

}

else

{

emptyupload.Text = "Choose your file";

}

}