****

**SQRA**

**MST Dry Weight Lookup**

|  |
| --- |
| **Technical Design Specification** |

**Document Information and Approvals**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version History** | |  | | |
| **Version #** | **Date** | | **Revised By** | **Reason for change** | | **Supervisor** |
| **1.0** | **8/14/2017** | | **Ashwani Chahal** | **Initial Documentation** | | **Curtis Hollingshead** |
|  |  | |  |  | |  |
|  |  | |  |  | |  |
|  |  | |  |  | |  |

**Table of Contents**

[1.0 Overview 4](#_Toc490490119)

[2.0 Architecture 4](#_Toc490490120)

[2.1 System Architecture 4](#_Toc490490121)

[2.2 Software Technologies 6](#_Toc490490122)

[2.4.1 Servers (.NET web server) 6](#_Toc490490123)

[2.4.2 Client (User Interface) 6](#_Toc490490124)

[3.0 Installation Instructions 6](#_Toc490490125)

[3.1 System Configuration & Tools Required 6](#_Toc490490126)

[3.2 Web Configuration 9](#_Toc490490127)

[4.0 Website Deployment 10](#_Toc490490128)

[5.0 Open Issues 11](#_Toc490490129)

[6.0 Point of Contact 11](#_Toc490490130)

1.0 Overview

This document provides an overview of the technical specifications of the webtool development for MST Dry Weight lookup.

This web tool provides a graphical user interface for a user to input Design IDs, fetch the data related to the MST Dry Weight, and display the results in form of a table.

The subsequent sections in this document form the basis for technical design, technical development, workflows, and procedures used during the development of the application(web-tool).

2.0 Architecture

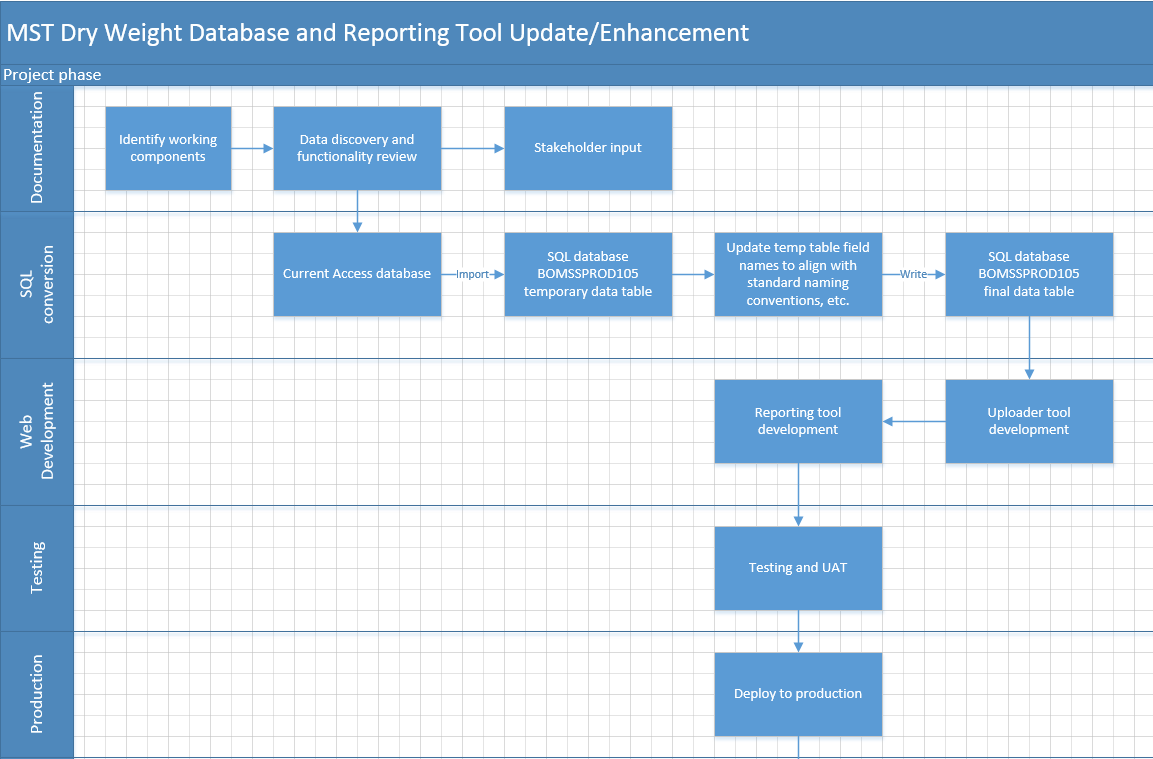
## 2.1 System Architecture

The system architecture consists of 3 parts – Front End, Back End, and Data System.

The Front End of the web tool is a graphical user interface that interacts with the user and displays the results.

The Back End of the web tool consists of a web server where all the web services reside. A web server accepts the user request (coming from the Front End), processes the request, sends the query to the Data System (using the data received in the query string from the Front End), and responds with the query result.

The Data System is a database server where our data reside. Query coming from the Web Server is processed by the Data System (Teradata in our case) and the query results are sent back to the Web Server.

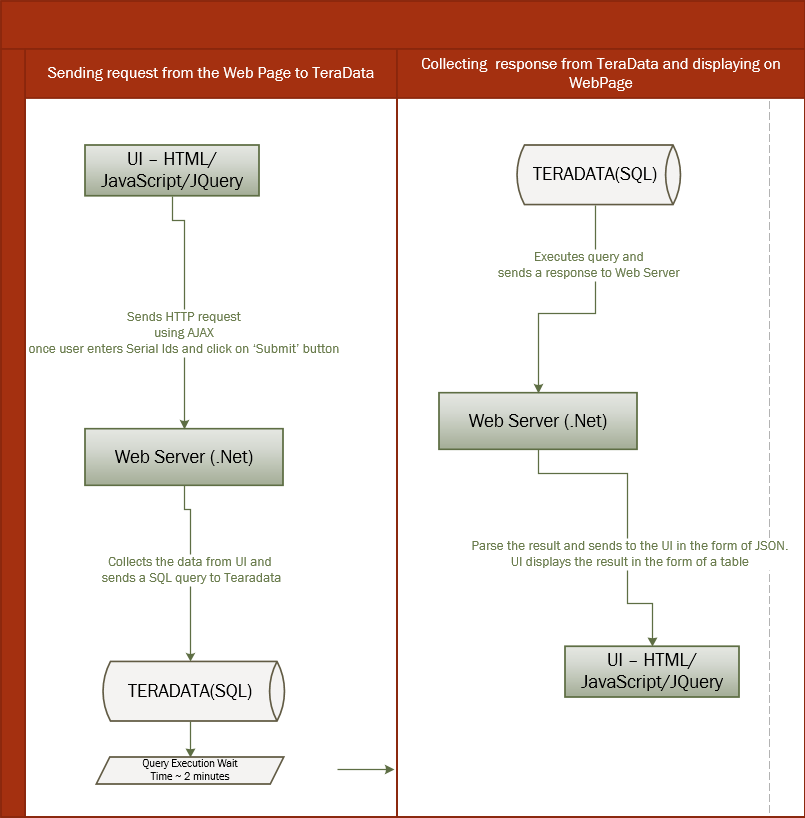


Below are the steps which occur in the mentioned order during the entire process:

1. User provides the input and click ‘submit’ button. This event creates an HTTP request (GET or POST) to send the data to web server
2. The web server accepts the request and uses the data, received in the HTTP request from front-end, to query Teradata
3. Teradata processes the query and responds with the result set to the web server
4. Web Server parses the data into JSON format and sends it to the front end
5. Data is received on the front end and is displayed in the form of a HTML Table

The historical data is present in the MS Access database present at the shared location:

[\\na.micron.com\root\micronweb\MTI\NT\webdata.micron.com\webapps\mfg\QA\packrel\Qual\_Data\MST\_Dry\_Weight.mdb](file:///\\na.micron.com\root\micronweb\MTI\NT\webdata.micron.com\webapps\mfg\QA\packrel\Qual_Data\MST_Dry_Weight.mdb)



## 2.2 Software Technologies

### 2.4.1 Servers (.NET web server)

A Web Server is created using ASP.NET framework 4.0

This server hosts the web services to receive and respond to the HTTP request from the client. This web server also contains the code to connect to the Teradata.

### 2.4.2 Client (User Interface)

A User Interface is created using HTML, CSS and JavaScript.

This interface contains the input box where user can provide the Design ID.

3.0 Installation Instructions

## 3.1 System Configuration & Tools Required

Entire code related to User Interface, Web Servers, and Teradata Query is updated at the shared repository of Micron - **BitBucket**. To clone the repo, use the below command in your GIT BASH Terminal:

**https://ACHAHAL@bitbucket.micron.com/bbdc/scm/~achahal/mst-dryweight.git**

**USER INTERFACE:**

Any text editor to create .html, .css, and .js files. I used Brackets (<http://brackets.io/>) to do the front-end coding as it provides a ‘live preview’ option.

An AJAX/JQuery GET Reuest is made from front end to the web srver:

**$("#loadingIndicator").show();**

**designID = designID.trim();**

**designID = "'" + designID + "'";**

**console.log(designID);**

**$.ajax({**

**type: 'GET',**

**cache: false,**

**timeout: 600000,**

**crossDomain: true,**

**url: 'http://webtest.micron.com/mstdryweight/RestServiceImpl.svc/Teradata/' + designID,**

**success: function (response) {**

**$("#loadingIndicator").hide();**

**var jsonData = response.fetchResultsResult;**

**if(response.fetchResultsResult.length <=2 ){alert('Design ID not found');}**

**console.log(jsonData);**

**$.each(JSON.parse(jsonData), function(k, v) {**

**var resultRow = "<tr class=\"text-center\"><td>" +**

**v.DESIGN\_ID + "</td><td>" +**

**v.NUMBER\_OF\_DIE\_IN\_PKG + "</td><td>" +**

**v.PACKAGE\_TYPE + "</td><td>" +**

**v.LEAD\_COUNT + "</td><td>" +**

**parseFloat(v.DRY\_WEIGHT).toFixed(3) + "</td></tr>";**

**$('#ResultTable tbody').append($(resultRow));**

**});**

**$('#demotable').show();**

**TERADATA:**

Teradata Studio Express is required to write a SQL query to the fetch the results. I created multiple CTEs (Common Table Expressions) to hold the result temporarily. SQL query (*mstQuery.sql)* is updated on bit bucket repository.

To make a connection to TeraData we require a .Net package – **Teradata.NET.Data.provider**

Below is the connection string to connect to TeraData and record results:

**//DATABASE CONNECTION .NET framework**

**TeraDataRetriever("BOTERAPROD09", "TERADATAREADER", "TeradataReader", "DataTera01");**

**public TeraDataRetriever(String dataSource, String Database, String Username, String Password)**

**{**

**TdConnectionStringBuilder conStrBuilder = new TdConnectionStringBuilder();**

**conStrBuilder.DataSource = dataSource;**

**conStrBuilder.Database = Database;**

**conStrBuilder.UserId = Username;**

**conStrBuilder.Password = Password;**

**conStrBuilder.AuthenticationMechanism = "LDAP";**

**Console.WriteLine("conn string was: " + conStrBuilder.ConnectionString);**

**con = new TdConnection**

**{**

**ConnectionString = conStrBuilder.ConnectionString**

**};**

**con.Open();**

**connectionError = false;**

**connectionErrorMessage = "";**

**}**

**var queryData = new DataTable();**

**var command = new TdCommand(query, con);**

**command.CommandTimeout = 120;**

**var adapter = new TdDataAdapter(command);**

**adapter.Fill(queryData);**

**MSACCES:**

We have also provided a link on the webpage to fetch the data from legacy MSAccess database. If user enters a design ID on the page related to MSAccess DB, the request reaches the webserver and it connects to MS DB before throwing the results back to the web client. The configuration string to connect to MS DB is written in the back-end code – MSAccessDataRetriever.cs.

**string connectionString;**

**connectionString = "Provider=Microsoft.ACE.OLEDB.12.0;";**

**connectionString = connectionString + @"Data Source=\\na.micron.com\root\micronweb\MTI\NT\webdata.micron.com\webapps\mfg\QA\packrel\Qual\_Data\MST\_Dry\_Weight.mdb;";**

**connectionString = connectionString + "Persist Security Info=True";**

**MSAccessDataRetriever mds = new MSAccessDataRetriever(connectionString);**

**public class MSAccessDataRetriever**

**{**

**//This is the MSACCESS connection object named con**

**OleDbConnection con;**

**public Boolean connectionError;**

**public String connectionErrorMessage;**

**//dataSource is provided here as an argument**

**public MSAccessDataRetriever(String connectionString)**

**{**

**try**

**{**

**con = new OleDbConnection(connectionString);**

**con.Open();**

**connectionError = false;**

**connectionErrorMessage = "Error: Failed to create a database connection";**

**}**

**catch (Exception e)**

**{**

**connectionError = true;**

**connectionErrorMessage = e.Message;**

**}**

**}**

**public String GetQueryResult(String query)**

**{**

**try**

**{**

**var queryData = new DataTable();**

**var command = new OleDbCommand(query, con);**

**command.CommandTimeout = 120;**

**var adapter = new OleDbDataAdapter(command);**

**adapter.Fill(queryData);**

**//USIING JsonConvert from Newtonsoft.Json**

**string myString = DataTableToJSONWithJSONNet(queryData);**

**return myString;**

**}**

**catch (Exception e)**

**{**

**return "{\"DataException\":\"" + e.Message + "\"}";**

**}**

**}**

**public string DataTableToJSONWithJSONNet(DataTable table)**

**{**

**string JSONString = string.Empty;**

**JSONString = JsonConvert.SerializeObject(table);**

**return JSONString;**

**}**

**WEB SERVER:**

Microsoft Visual Studio Professional 2015 is used to create Web Services.

Go to: File -> New -> Project

Search: WCF

Click on: WCF Service Application – Visual C#

Create a new WCF application which contains web services for http requests. To know how to create a new WCF service application, please refer to this link:

<https://www.codeproject.com/Articles/105273/Create-RESTful-WCF-Service-API-Step-By-Step-Guide>

To install the package, right click on your solution in visual studio and select ‘manage NuGet packages’. Browse for a package and install.

To convert table format data, returned by TeraData, into JSON format is – **Newtonsoft.Json** which provides a function JsonConvert to convert a data table into JSON.

public string DataTableToJSONWithJSONNet(DataTable table)

{

string JSONString = string.Empty;

JSONString = JsonConvert.SerializeObject(table);

return JSONString;

}

**3 WAYS TO CONVERT A DATA TABLE TO JSON:**

http://www.c-sharpcorner.com/UploadFile/9bff34/3-ways-to-convert-datatable-to-json-string-in-Asp-Net-C-Sharp/

To create a RESTful service, we need a WCF service application (like explained above) then we must make certain changes in Service interface and web configuration file (*web.config*).

## 3.2 Web Configuration

There is a web.config file in the solution folder which contains the website configuration. Few important changes are mentioned below:

**<system.web>**

**<compilation debug="true" targetFramework="4.0"/>**

**<httpRuntime maxQueryStringLength="32768" maxUrlLength="65536" relaxedUrlToFileSystemMapping="true" targetFramework="4.0"/>**

**</system.web>**

**<services>**

**<service name=" mstDryWeight.RestServiceImpl" behaviorConfiguration="ServiceBehaviour">**

**<endpoint address="" binding="webHttpBinding" contract="mstDryWeight.IRestServiceImpl" behaviorConfiguration="web"/>**

**</service>**

**</services>**

**<protocolMapping>**

**<add binding="basicHttpsBinding" scheme="https"/>**

**</protocolMapping>**

**<serviceHostingEnvironment aspNetCompatibilityEnabled="false" multipleSiteBindingsEnabled="true"/>**

**<system.webServer>**

**<modules runAllManagedModulesForAllRequests="true"/>**

**<security>**

**<requestFiltering>**

**<requestLimits maxQueryString="32768">**

**</requestLimits>**

**</requestFiltering>**

**</security>**

**<httpProtocol>**

**<customHeaders>**

**<add name="Access-Control-Allow-Origin" value="\*" />**

**<add name="Access-Control-Allow-Headers" value="Content-Type" />**

**<add name="Access-Control-Allow-Methods" value="GET, POST, PUT, DELETE, OPTIONS" />**

**</customHeaders>**

**</httpProtocol>**

**<directoryBrowse enabled="true"/>**

**</system.webServer>**

4.0 Website Deployment

Once ready to be deployed, a request was raised with webcast team at below location:

<http://collab.micron.com/is/sites/webcenter/_layouts/15/WopiFrame2.aspx?sourcedoc=/is/sites/webcenter/Secure%20Web%20Environment/How%20to%20Submit%20Web%20Hosting%20Request%20in%20Secure%20Web%20Environment.docx&action=default&DefaultItemOpen=1>

Support group: MST\_DRY\_WEIGHT\_WEBCONTACT

Target Group: MST\_DRY\_WEIGHT\_WEBFORM

The hosting of “mstdryweight” web application in Boise Secure Web environment was completed successfully.

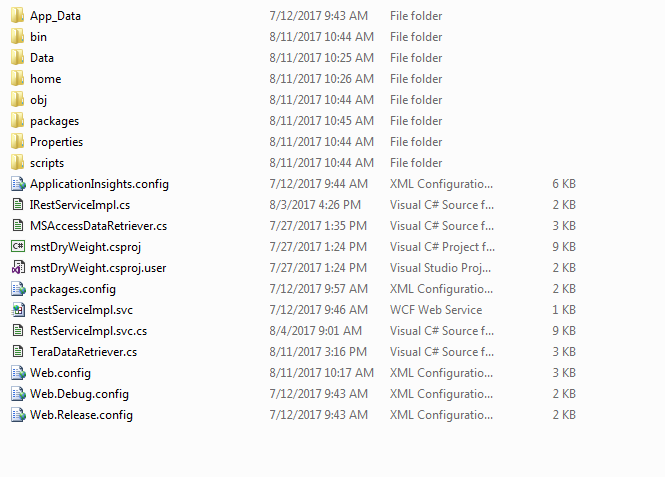
Below are the Test and Production web links for the same:

TEST: http://webtest.micron.com/mstdryweight/

PROD: http://web.micron.com/mstdryweight/

Copy all the backend and frontend files to - \\bowebfs01\bowebtest\mstdryweight\ and perform testing.

This is how the folder layout looks after placing the relevant files:



**Home folder contains index.html (which is the user interface)**

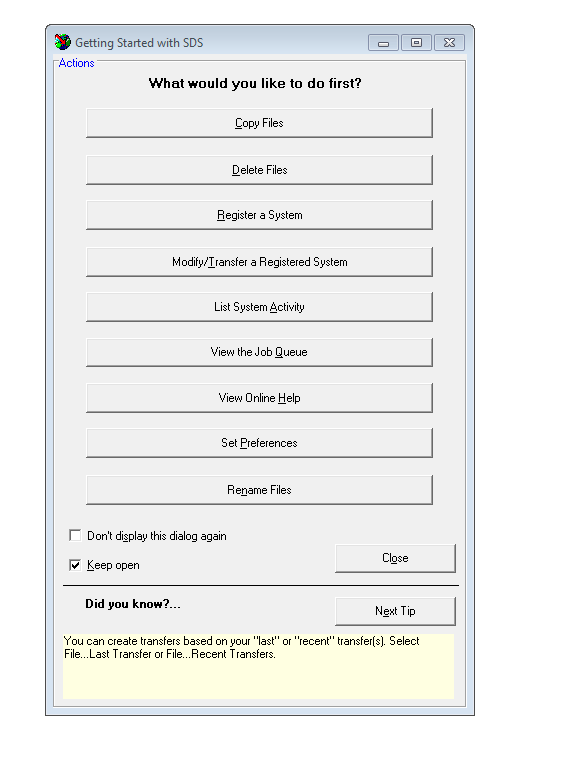
**Data folder contains the images, style, and JavaScript for the user interface.**

**Rest all other files are copied from the Visual Studio Solution for web server (back end)**

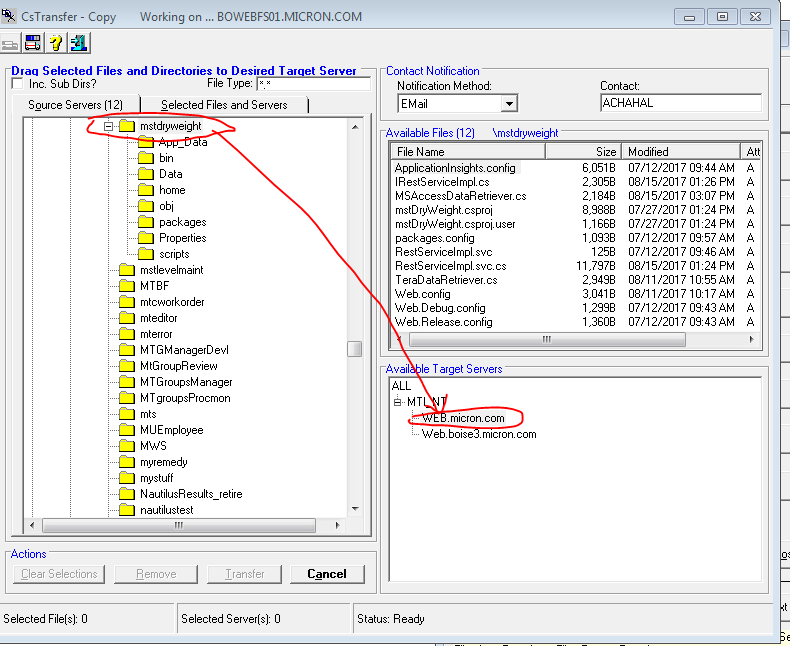
**In the url paste** <http://webtest.micron.com/mstdryweight/home>

**Once testing is done, deploy the website to production as explained below:**

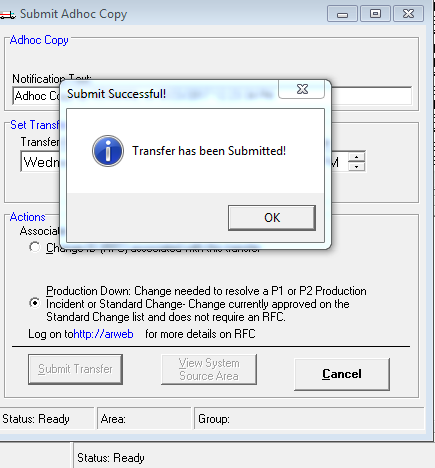
1. Install SDS from SAM
2. Right click SDS and run as administrator
3. You’ll see the below screen:



1. Click on ‘copy files’
2. Go to MTI -> NT -> BOWWEBTEST.micron.com -> mstdryweight
3. Drag the entire folder to ‘available target servers’ - > WEB.micron.com and then click on **Transfer**



1. Select the second radio button and ‘submit transfer’



1. The files will be copied on - [\\bowebfs01\bowebprod\mstdryweight](file:///\\bowebfs01\bowebprod\mstdryweight)
2. Go to <http://web.micron.com/mstdryweight/>home to access the website on production

5.0 Open Issues

No open issues.

6.0 Point of Contact

**Web Team: Dilip Kumar Pal (**[dpal@micron.com](mailto:dpal@micron.com)**)**

**Craig Kickel (**[cskickel@micron.com](mailto:cskickel@micron.com)**)**