This solution will work for you. It is quite complicated as Talend solutions go, so I will give you a step by step guide. You will need to acquire and use third party libraries. I will point you to these, but you will need to configure them. Information can be found on this on the forum.

1) Create a routine called WebScraperUtilities

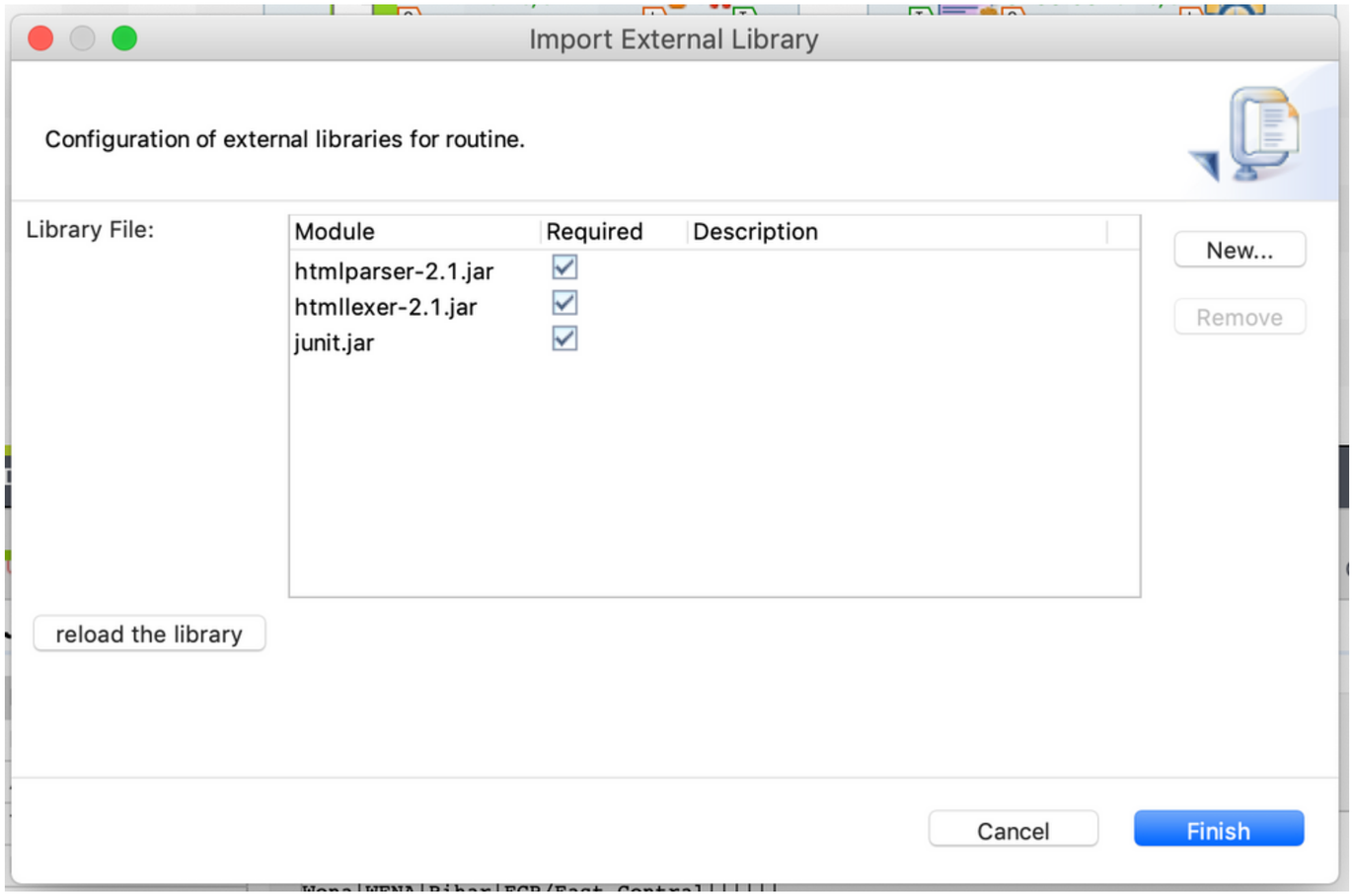
Your routine should look like this.....

```
package routines;

import java.util.ArrayList;

import org.htmlparser.Parser;
import org.htmlparser.util.NodeList;
import org.htmlparser.util.ParserException;
import org.htmlparser.tags.TableHeader;
import org.htmlparser.tags.TableRow;
import org.htmlparser.tags.TableColumn;
import org.htmlparser.visitors.NodeVisitor;
import org.htmlparser.Tag;
import org.htmlparser.tags.TableTag;


/*
 * user specification: the function's comment should contain keys as follows: 1. write about the function's comment.but
 * it must be before the "{talendTypes}" key.
 *
 * 2. {talendTypes} 's value must be talend Type, it is required . its value should be one of: String, char | Character,
 * long | Long, int | Integer, boolean | Boolean, byte | Byte, Date, double | Double, float | Float, Object, short |
 * Short
 *
 * 3. {Category} define a category for the Function. it is required. its value is user-defined .
 *
 * 4. {param} 's format is: {param} <type>[(<default value or closed list values>)] <name>[ : <comment>]
 *
 * <type> 's value should be one of: string, int, list, double, object, boolean, long, char, date. <name>'s value is the
 * Function's parameter name. the {param} is optional. so if you the Function without the parameters. the {param} don't
 * added. you can have many parameters for the Function.
 *
 * 5. {example} gives a example for the Function. it is optional.
 */
public class WebScraperUtilities {

    /**
     * parseTable: a method to process <table> data on a website.
     * This method can easily be extended if requirements are different for other website tables.
     *
     *
     * {talendTypes} ArrayList
     *
     * {Category} User Defined
     *
     * {param} String("html") input: A string representation of the website
     *
```

```
 */
public static ArrayList<ArrayList<String>> parseTable(String html){

    //Define an ArrayList to hold the data found by the NodeVisitor class
    final ArrayList<ArrayList<String>> rows = new ArrayList<ArrayList<String>>();

    //if the html string is not null
    if(html!=null){

        //Instantiate an instance of the NodeVisitor class and override its visitTag method
        final NodeVisitor linkVisitor = new NodeVisitor() {

            @Override
            public void visitTag(Tag tag) {

                //If the tag is an instance of the TableTag, process the data
                if(tag instanceof TableTag ){
                    TableTag tt = (TableTag)tag;
                    TableRow[] trows = tt.getRows();

                    //Used to identifiy that the table header row has been found
                    boolean headersFound = false;

                    //For each row, get the column data
                    for(int i = 0; i<trows.length; i++){
                        TableRow tr = trows[i];
                        TableColumn[] tcols = tr.getColumns();

 //If the headers have not been found for this table, try this
 if(!headersFound){
  TableHeader[] ths = tr.getHeaders();

  ArrayList<String> tmpHeader = new ArrayList<String>();
  if(ths.length>0){

   for(int x = 0; x<ths.length; x++){
    tmpHeader.add(ths[x].toPlainTextString());
   }

  }
  //Add the header row to the output ArrayList
  rows.add(tmpHeader);
  headersFound = true;
}

 ArrayList<String> tmpRow = new ArrayList<String>();


 //if this row has more than 1 column
 if(tcols.length>1){
     //Get each column value
     for(int x = 0; x<tcols.length; x++){
         TableColumn tc = tcols[x];
         String columnVal = tc.toPlainTextString().trim();

         //Remove " " strings from the column values
```

```
                            columnVal = columnVal.replaceAll(" ", "");
                            tmpRow.add(columnVal);
                        }
                        rows.add(tmpRow);
                    }
                }
            }
        };


        //Instantiate an instance of the Parse class
        Parser parser = Parser.createParser(html, null);

        //Instantiate an empty instance of the NodeList class
        NodeList list;

        try {
            //Assign the NodeList instance (list) an object
            list = parser.parse(null);
            //Assign the list object a Visitor ....defined above.
            list.visitAllNodesWith(linkVisitor);
            return rows;//list.toHtml();
        } catch (ParserException e) {
            // Could not parse HTML, return original HTML
            return null;
        }
    }else{
            return rows;
        }

    }

}
```

You will need to acquire the following libraries.....



The screen above shows the "Edit Routine Libraries" window that comes up when you right click on the routine you have just created. The routine must be saved and closed before you do this. The JUnit library is already available with Talend. The other two libraries can be downloaded here...

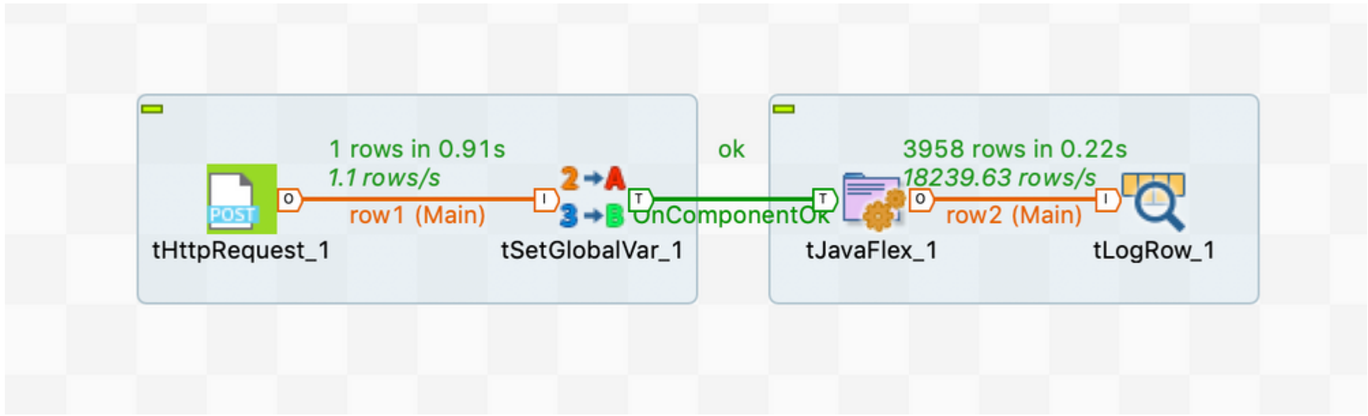
https://mvnrepository.com/artifact/org.htmlparser/htmlparser/2.1

https://mvnrepository.com/artifact/org.htmlparser/htmllexer/2.1

You will need to add them as modules to Talend.

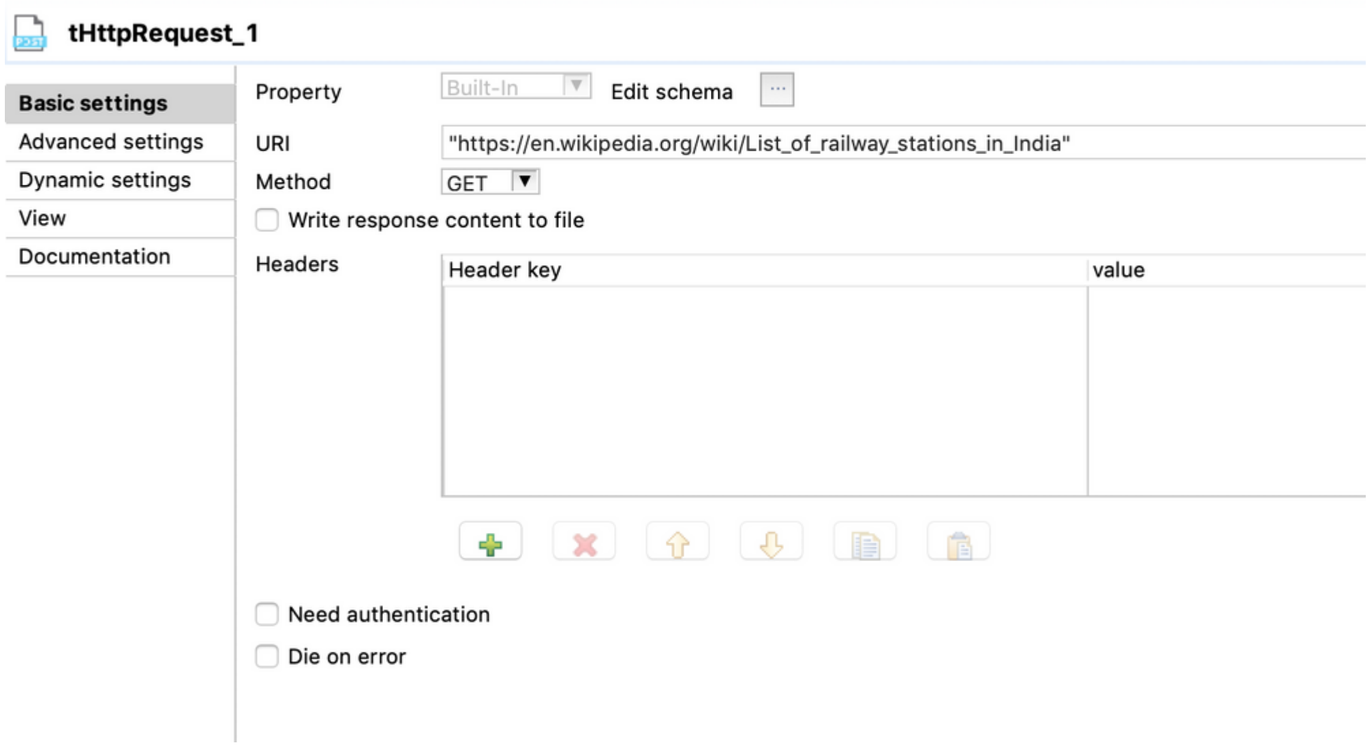
2) Create a job to use the routine

The job shown below is a basic job that will parse your webpage and output the table content to the output window. You will need to modify it to meet your requirements.



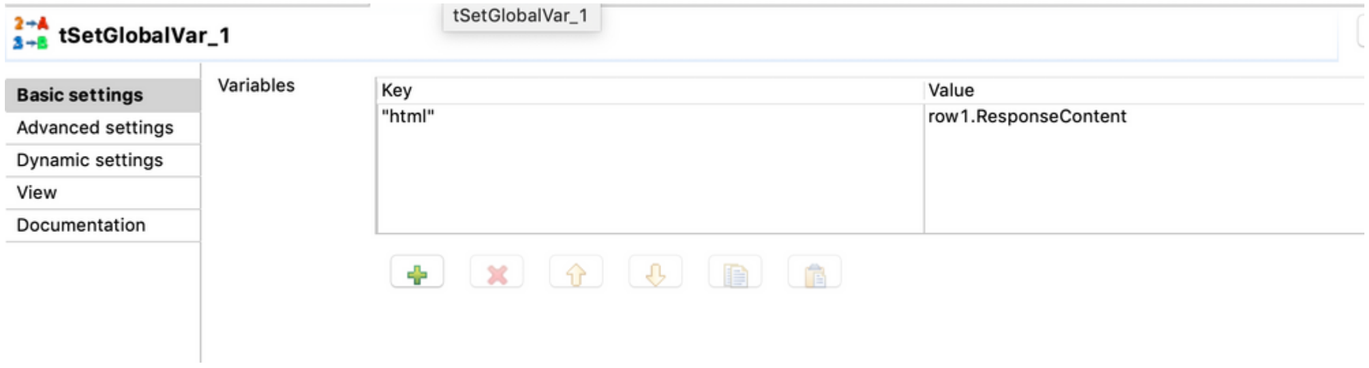
2a) The tHttpRequest

This component simply reads your HTML page. It's configuration looks like below...



2b) The tSetGlobalVar

This component is simply used to store the HTML String in memory. It is configured as below....



2c) The tJavaFlex component

This is where the magic happens. This component has 6 output String columns configured in the schema (one for each of the columns in the tables). It also has some code in 3 code sections. These will be shown below....

Start Code

```
// start part of your Java code
java.util.ArrayList<java.util.ArrayList<String>> list = routines.WebScraperUtilities.parseTable((String)globalMap.get("html"));
java.util.Iterator<java.util.ArrayList<String>> it = list.iterator();

while(it.hasNext()){
```

Main Code

```
// here is the main part of the component,
// a piece of code executed in the row
// loop

java.util.ArrayList<String> columns = it.next();

try{
        row2.col1 = columns.get(0);
        row2.col2 = columns.get(1);
        row2.col3 = columns.get(2);
        row2.col4 = columns.get(3);
        row2.col5 = columns.get(4);
        row2.col6 = columns.get(5);

    }catch(java.lang.IndexOutOfBoundsException iobe){
          System.out.println(iobe.toString());
     }
```

End Code

```
// end of the component, outside/closing the loop
}
```

Essentially, the above is a while loop broken into 3 parts; the start of the loop, the loop contents and the end of the loop. The start calls the routine you built above and sends it the HTML String. That routine returns an ArrayList which is full of your table data. The middle part of this (Main Code) returns each row in the ArrayList as a Talend data row.

2d) The tLogRow component

This needs no explanation. It is just used to print the data to the output window.

When you run this, assuming everything has been done correctly, you will see this (a small sample) in the output window....

```
Station Name|Station Code|State|Railway Zone|Elevation|Notes

Abada|ABB|West Bengal|SER/South Eastern|7 m|&#91;1&#93;
Abhaipur|AHA|Bihar|ER/Eastern|52 m|&#91;2&#93;
Abhayapuri|AYU|Assam|NFR/Northeast Frontier|45 m|&#91;3&#93;
Abohar|ABS|Punjab|NR/Northern|186 m|&#91;4&#93;
Abu Road|ABR|Rajasthan|NWR/North Western|260 m|&#91;5&#93;
Abutara Halt|ABW|West Bengal|NFR/Northeast Frontier Railway|36 m|
Acharapakkam|ACK|Tamil Nadu|SR/Southern|39 m|&#91;6&#93;
Acharya Narendra Dev|ACND|Uttar Pradesh|Northern|104 m|
Achalganj|ACH|Uttar Pradesh|NR/Northern|133 m|&#91;7&#93;
Achalpur|ELP|Maharashtra|CR/Central|388 m|&#91;8&#93;
Achhalda|ULD|Uttar Pradesh|NCR/North Central|147 m|&#91;9&#93;
Achhnera Junction|AH|Uttar Pradesh|NCR/North Central|170 m|
Adarshnagar|AHO|Rajasthan|NWR/North Western|472 m|
Adarshnagar Delhi|ANDI|Delhi|NR/Northern|215 m|
Adas Road|ADD|Gujarat|WR/Western|42 m|
Adesar|AAR|Gujarat|WR/Western|35 m|
Adgaon Buzurg|ABZ|Maharashtra|SCR/South Central|310 m|
Adilabad|ADB|Telangana|SCR/South Central|248 m|
Adipur|AI|Gujarat|WR/Western|35 m|
```