8/13/2018

Jagacy Software
www.jagacy.com

Jagacy Total Access

Total Access is a powerful ETL data processing tool and report generator written in 100% Java -- it runs on any platform. It leverages the power of an enhanced awk scripting language that specializes in data conversion and management, cutting development time to a fraction of existing methods. Total Access can be used for data mining, generating adhoc analytics reports, and manipulating data from a variety of data sources (including databases, XML and Microsoft® Excel™ spreadsheets). In addition, it is Web enabled, providing online report generation and viewing.

Table of Contents

1. Introduction

Total Access is based on the awk programming language (please see https://www.gnu.org/software/gawk/manual/html_node/Getting-Started.html#Getting-Started, but disregard the gawk-specific items). It enhances the language by providing a rich set of additional functions, and by allowing data other than text files to be processed. If the Input Type (IT) variable is set to “database”, “excel”, “xml” or “csv”, Total Access reads databases, xls/xlsx, XML or CSV files, respectively. Each row is passed to the script as a line, with each column/field separated by the value of the Input Field Separator (IFS) variable. Please refer to the examples in the Total Access installation directory for more information.

2. Data Mining Example

The following Total Access script determines which people live in an area where given houses are for sale. The customer’s information is stored in a database table while the houses for sale are stored in an excel spreadsheet. The database information is stored in an XML properties file (but could alternatively be specified in the script):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
      <comment>customer_sql.xml: customer table query sql</comment>

      <entry key="driverClass">org.hsqldb.jdbc.JDBCDriver</entry>
      <entry key="databaseUrl">jdbc:hsqldb:file:testdb</entry>
      <entry key="databaseUserName">SA</entry>
      <entry key="databasePassword"></entry>

      <entry key="sqlQuery">
      <![CDATA[
            SELECT * FROM customer
            WHERE zip = :zip
      ]]>
      </entry>

</properties>
```

The Total Access script that performs the data mining is as follows:

```
# excelDbExample.awk:
BEGIN   {

        # These can be specified on the command line.
        IT = "excel:sheet=real_estate";
        FS = "[\t]";

        loadArray(querySql, "customer_sql.xml");
```

```
        # Use sorted array.
        createArray(zips, 0);
}

# Skip headers
(FNR == 1)  {
        next;
}

# Check for empty line
($0 == "null") {
        next;
}

# else
{
        zips[$3]++;
}

END {
        # Tell compiler that this is an array.
        customer[0];

        # Use sorted array.
        createArray(customer, 0);

        # Find database entries with zip codes in the excel file.
        for (zip in zips) {
            querySql[":zip"] = integer(zip);
            statementHandle = queryDatabase(querySql, outArrays);
            for (i in outArrays) {
                copyArray(customer, outArrays[i]);
                print customer;
            }
        }

        if (statementHandle != "") {
            closeDatabaseStatement(statementHandle);
        }
}
```

This script can be executed from Windows as follows:

```
java -cp totalAccess.jar;hsqldb.jar com.jagacy.totalAccess.Main -f
excelDbExample.awk real_estate_full.xlsx
```

or from Mac/Linux:

```
java -cp totalAccess.jar:hsqldb.jar com.jagacy.totalAccess.Main -f
excelDbExample.awk real_estate_full.xlsx
```

3. Report Generation Web Servlet Example

Total Access can generate adhoc reports from within a Web application for viewing online:

```
package com.jagacy.totalAccess.servlet;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.jagacy.totalAccess.Main;

/*************************************************************
 *
 * An example excel servlet.
 *
 *
 * To use this in your web application:
 *
 * 1) Copy totalAccess.jar and this file into your web application.
 * 2) Copy csv2excel.awk and real_estate_basic.csv to the
 *    WEB-INF/totalAccess directory in your web application.
 * 3) Add the following to web.xml:
 *         <servlet>
 *              <servlet-name>csv2excel</servlet-name>
 *              <servlet-class>
 *                  com.jagacy.totalAccess.servlet.ExcelServlet
 *              </servlet-class>
 *          </servlet>
 *
 *          <servlet-mapping>
 *              <servlet-name>csv2excel</servlet-name>
 *              <url-pattern>/csv2excel</url-pattern>
 *          </servlet-mapping>
 *
 * 4) Start your web application.
 * 5) Type one of the following into your web browser:
 *      http://localhost:8080/<webapp>/csv2excel?ext=xls
 *      http://localhost:8080/<webapp>/csv2excel?ext=xlsx
 */

public class ExcelServlet extends HttpServlet {
    private static final long serialVersionUID = -2122149597586470901L;

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException,
            IOException {
        BufferedInputStream input;
        BufferedOutputStream output;
        File forwardFile;
        String forwardName;
        boolean attachment;
```

```
input = null;
output = null;
attachment = false;

String ext = request.getParameter("ext");

if (!"xls".equals(ext) && !"xlsx".equals(ext)) {
    ext = "xls";
}

forwardFile = File.createTempFile("report_", "." + ext);
forwardFile.deleteOnExit();

forwardName = forwardFile.getCanonicalPath();

String path = request.getServletContext().getRealPath(
    "/WEB-INF/totalAccess");

String[] params = new String[] {"-f", path + "/csv2excel.awk",
        "-v", "excelFileName=" + forwardName,
        path + "/real_estate_basic.csv"
};

try {
    new Main().invoke(params);
} catch (Throwable t) {
    throw new RuntimeException(t);
}

try {
    // Prepare.
    input = new BufferedInputStream(
        new FileInputStream(forwardName));
    int contentLength = input.available();
    String fileName = "real_estate_basic." + ext;

    String contentType = "application/vnd.ms-excel";
    if (fileName.endsWith(".xlsx")) {
        contentType = "application/vnd.openxmlformats-" +
            "officedocument.spreadsheetml.sheet";
    }

    String disposition = attachment ? "attachment" : "inline";

    // Init servlet response.
    response.setHeader("cache-control", "no-cache, max-age=0");
    response.setHeader("pragma", "no-cache");
    response.setHeader("expires", "-1");
    response.setDateHeader("Last-Modified",
        System.currentTimeMillis());
    //response.setContentLength(contentLength);
    response.setContentType(contentType);
    response.setHeader("Content-disposition", disposition
            + "; filename=\"" + fileName + "\"");
    output = new
        BufferedOutputStream(response.getOutputStream());

    // Write file contents to response.
```

```
            while (contentLength-- > 0) {
                output.write(input.read());
            }

            // Finalize task.
            output.flush();
        } finally {
            if (input != null) {
                try {
                    input.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (output != null) {
                try {
                    output.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

4. Differences from awk

String literals	Supports \t, \n, \r, \b, \f, \", \', \\, \a, \v, \### (octal), \u#### (unicode), \x## (hexadecimal).
Number literals	Supports 0### (octal), 0x##, or 0x#### (hexadecimal).
Regular expressions	Of the form: https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#sum
FS variable	If set to 1 space character (" "), matches on space (" "), \t, \n, \r, \f.
CONVFMT variable, printf(), sprintf()	Of the form: https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html#syntax
getline, getline < "file"	Subject to IT, IFS variable processing.
variables	Also supports Nodes and integers, but does not support strnums.

5. Variables

IT (Input Type) – Allows Total Access to read files other than text files. It can be specified in the script or on the command line (to change input types). Please

see csv2db.bat (or csv2db.sh for Mac/Linux) for more information. Acceptable values are:

- excel – The default sheet name is “Sheet1”, booleans are not converted to 0 or 1, an empty row is returned as “null”, and an empty column is returned as an empty string (“”).

- excel:[sheet=<sheetName>,awkBoolean=<true|false>,emptyRow=<value>,emptyColumn=<value>]

- csv – The default delimiter is comma (“,”), an empty row is returned as “null”, and an empty column is returned as an empty string (“”).

- csv:[delimiter=<value>,emptyRow=<value>,emptyColumn=<value>]

- database – Reads from the database specified in the given XML properties file (please see Database XML Properties).

- database:[*properties or :parameters*]

- xml – Uses a forward slash (“/”) as the default xmlQuery XPath (please see http://en.wikipedia.org/wiki/XPath). Does not normalize a node (that is, combine text and CDATA sections), and does not include the xml header nor indent a node’s XML when the node is converted into a string.

  The variable _**$NODE** contains each node read, and **$0** is set to:

  <node name><IFS><attr1>=<value1><IFS><attr2>=<value2><IFS>…

- xml:[xmlQuery=<value>,normalize=<true|false>,xmlHeader=<true|false>,xmlIndent=<true|false>]

- text (the default).

IFS (Input Field Separator) – Used to separate columns/fields in database query results, Excel, XML and CSV input files. The default is tab (“\t”).

_$NODE – The current node read from an XML file. The default is a NUL node.

DATEFMT – The format used for dates. Of the form: https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html. The default is "yyyy-MM-dd HH:mm:ss.SSS".

SYSPROPS – The Java system properties as an associative array.

6. Array Functions

- **copyArray**(target, source)

  Copies the source array to the target array. Returns the target array.

- **createArray**(array, dimension, [subscript1, subscript2, …, subscriptN, value, …])

  Creates a sorted array. If dimension is 0, consecutive integer values are used as subscripts (of the form 1,2,3,4…), indexed from 1. Otherwise, the subscript argument(s) are used. Returns the array.

- **clearArray**(array)

  Rather than the delete statement, this function just clears the array for reuse. Returns the array.

- **mergeArray**(target, source1, [source2, source3, …])

  Merges source1, source2, etc. into the target array. Returns the target array.

- len = **arrayLength**(array)

  Returns the number of elements in the array.

- **subArray**(target, subscript, source)

  Copies all source array elements that have the initial subscript to the target array. Returns the target array.

- **loadArray**(target, propertiesFile)

  Loads the array from the specified properties file (if the file ends with .properties) or XML properties file (if the file ends with .xml). Returns the array.

7. Data Type Functions

- type = **typeof**(var)

  Returns “AssocArray”, “Double”, “Integer”, “Node" or “String”.

- str = **string**(var)

  Returns the string value of var.

- dbl = **double**(var)

  Returns the double value of var, or 0 if it cannot be converted to a double.

- int = **integer**(var)

  Returns the integer value of var, or 0 if it cannot be converted to an integer. Any fraction is truncated.

- numeric = **isNumeric**(numericStr)

  Returns 1 if string is numeric; 0 otherwise.

8. Date Functions

- dateStr = **date**([format])

  Returns the current date/time as a string. If format isn't specified, the value of the DATEFMT variable is used.

- dateStr = **convertDate**(fromDate, fromFormat, [toFormat])

  Converts the fromDate with the format fromFormat to toFormat. If toFormat isn't specified, the value of the DATEFMT variable is used.

9. File Functions

- exists = **fileExists**(fileName)

  Returns 1 if the file exists, 0 otherwise.

- deleted = **deleteFile**(fileName)

  Deletes the file with the name fileName. Returns 1 if the file was deleted, 0 otherwise.

- renamed = **renameFile**(fromName, toName)

Renames fromName to toName. Returns 1 if the file was renamed, 0 otherwise.

- created = **mkdir**(dirName)

  Makes the directory dirName. Returns 1 if the directory was created, 0 otherwise.

10. String Functions

- unescapedStr = **unescapeString**(escapedStr)

  Returns a string with all special characters (preceded by "\") unescaped.

- escapedStr = **escapeString**(unescapedStr)

  Returns a string with all special characters escaped (with "\").

- trimmedStr = **trim**(untrimmedStr)

  Returns a string with all beginning and ending whitespace removed.

11. Excel Functions

In addition to reading xls/xlsx files using the IT variable, Total Access provides the following functions. Please see the csv2excel.awk and db2excel.awk examples in the Total Access installation directory for more information:

- handle = **openExcel**(fileName, [sheetName])

  Opens a new or existing Excel spreadsheet, using the sheet sheetName. If the spreadsheet is new, the extension of fileName (.xls or .xlsx) determines what type of spreadsheet will be created. If sheetName is not specified, it defaults to "Sheet1". Returns a handle for all subsequent operations.

- **createExcelFont**(handle, fontId, fontStyles)

  Fonts are a limited resource, and should be reused. This function creates an Excel font with the name fontId, and applies the fontStyles to it (please see Excel Font Styles). The handle is the value returned by openExcel().

- **createExcelStyle**(handle, styleId, cellStyles)

  Cell styles are also a limited resource (3000), and should be reused. This function creates an Excel cell style and applies the cellStyles to it (please see Excel Cell Styles). The handle is the value returned by openExcel().

- **printExcelValue**(handle, row, column, value, [settings])

  Stores the value in the cell specified by row and column, applying the cell settings (please see Excel Cell Settings). Row and column are indexed from 1. The handle is the value returned by openExcel().

- **printExcelRTF**(handle, row, column, rtfStr, [settings])

  Stores the Rich Text Format (RTF) string in the cell specified by row and column, applying the cell settings (please see Excel Cell Settings). Row and column are indexed from 1. The handle is the value returned by openExcel().

- **printExcelDate**(handle, row, column, dateStr, [settings], [dateFmt])

  Converts dateStr using dateFmt (or the value of the DATEFMT variable if dateFmt is not specified), and stores it in the cell specified by row and column, applying the cell settings (please see Excel Cell Settings). Row and column are indexed from 1. The handle is the value returned by openExcel().

- **printExcelFormula**(handle, row, column, formula, [settings])

  Stores the formula in the cell specified by row and column, applying the cell settings (please see Excel Cell Settings). Row and column are indexed from 1. The handle is the value returned by openExcel().

- **printExcelLink**(handle, row, column, value, link, [settings])

  Stores the hyperlink in the cell specified by row and column, applying the cell settings (please see Excel Cell Settings). Row and column are indexed from 1. The handle is the value returned by openExcel().

- **printExcelBoolean**(handle, row, column, value, [settings])

  Stores the boolean in the cell specified by row and column, applying the cell settings (please see Excel Cell Settings). Row and column are indexed from 1. The handle is the value returned by openExcel().

- rowCount = **getExcelRowCount**(handle)

Returns the number of rows in the sheet. The handle is the value returned by openExcel().

- columnCount = **getExcelColumnCount**(handle, row)

  Returns the number of columns in a row. Row is indexed from 1. The handle is the value returned by openExcel().

- value = **getExcelValue**(handle, row, column, [toAwkBoolean], [emptyRowValue], [emptyColumnValue])

  Returns the value of the cell at the row and column specified. If toAwkBoolean is 1, booleans are returned as 0 or 1. If emptyRowValue is set, this value will be returned for empty rows (otherwise "null" is returned). If emptyColumnValue is set, this value will be returned for empty cells (otherwise empty string ("") is returned). Row and column are indexed from 1. The handle is the value returned by openExcel().

- sheetName = **getExcelSheetName**(handle)

  Returns the sheet name being processed. The handle is the value returned by openExcel().

- line = **getExcelLine**(handle, [toAwkBoolean], [emptyRowValue], [emptyColumnValue], [outFields])

  Returns the next row in an Excel sheet as a line. Cells in the line are separated by the value of the IFS variable. Cells are also stored in the sorted outFields array (if specified), with each entry indexed with a consecutive integer (starting at 1). If toAwkBoolean is 1, booleans are returned as 0 or 1. If emptyRowValue is set, this value will be returned for empty rows (otherwise "null" is returned). If emptyColumnValue is set, this value will be returned for empty cells (otherwise empty string ("") is returned). A NUL string ("\0") is returned when no more rows exist. The handle is the value returned by openExcel().

- **closeExcel**(handle, [closeSettings])

  Closes the Excel sheet, applying the closeSettings (please see Excel Close Settings).). The handle is the value returned by openExcel().

12. Excel Font Styles

Font Style	**Possible Values**
fontColor	See Excel Colors

bold	0 or 1
italic	0 or 1
strikeout	0 or 1
underline	none single double singleAccounting doubleAccounting
fontName	A valid font name.
fontPointSize	A valid point size.

13. Excel Cell Styles

Cell Style	**Possible Values**
useFont	A font id created with createExcelFont()
halign	center left right justify general
valign	center bottom top justify distributed
wrapText	0 or 1

dataFormat	Either a number:
	`0, "General"`
	`1, "0"`
	`2, "0.00"`
	`3, "#,##0"`
	`4, "#,##0.00"`
	`5, "$#,##0_);($#,##0)"`
	`6, "$#,##0_);[Red]($#,##0)"`
	`7, "$#,##0.00);($#,##0.00)"`
	`8, "$#,##0.00_);[Red]($#,##0.00)"`
	`9, "0%"`
	`0xa, "0.00%"`
	`0xb, "0.00E+00"`
	`0xc, "# ?/?"`
	`0xd, "# ??/??"`
	`0xe, "m/d/yy"`
	`0xf, "d-mmm-yy"`
	`0x10, "d-mmm"`
	`0x11, "mmm-yy"`
	`0x12, "h:mm AM/PM"`
	`0x13, "h:mm:ss AM/PM"`
	`0x14, "h:mm"`
	`0x15, "h:mm:ss"`
	`0x16, "m/d/yy h:mm"`

	// 0x17 - 0x24 reserved for international and undocumented 0x25, "#,##0_);(#,##0)" 0x26, "#,##0_);[Red](#,##0)" 0x27, "#,##0.00_);(#,##0.00)" 0x28, "#,##0.00_);[Red](#,##0.00)" 0x29, "_(* #,##0_);_(* (#,##0);_(* \"-\"_);_(@_)" 0x2a, "_($* #,##0_);_($* (#,##0);_($* \"-\"_);_(@_)" 0x2b, "_(* #,##0.00_);_(* (#,##0.00);_(* \"-\"??_);_(@_)" 0x2c, "_($* #,##0.00_);_($* (#,##0.00);_($* \"-\"??_);_(@_)" 0x2d, "mm:ss" 0x2e, "[h]:mm:ss" 0x2f, "mm:ss.0" 0x30, "##0.0E+0" 0x31, "@" - This is text format. 0x31 "text" - Alias for "@" Or a custom string format (please see https://support.microsoft.com/en-us/help/264372/how-to-control-and-understand-settings-in-the-format-cells-dialog-box)
_fillForegroundColor	(Note: Set foreground color before background color) See Excel Colors
fillBackgroundColor	(Note: Set foreground color before background color) See Excel Colors
fillPattern	altBars, bigSpots, bricks, diamonds, fineDots, leastDots, lessDots, noFill, solidForeground, sparseDots, thickBackwardDialog, thickForwardDialog, thinHorzBands, thinVertBands
borderBottom, borderLeft, borderRight, borderTop	dashDot, dashDotDot, dashed, dotted, double, hair, medium, mediumDashDot, mediumDashed, none, slantedDashDot, thick, thin
borderBottomColor, borderLeftColor, borderRightColor, borderTopColor	See Excel Colors
shrinkToFit	0 or 1
quotePrefix	0 or 1
locked	0 or 1
hidden	0 or 1
rotation	A valid rotation value.
indentation	A valid indentation value.

14. Excel Cell Settings

Cell Settings	Possible Values
useCellStyle	A style id created with createExcelStyle()
activeCell	1

15. Excel Close Settings

Close Settings	Possible Values
selectedSheet	1
activeSheet	1
hiddenSheet	1
autoSizeColumn, <*columnNumber*>	1. Note that columnNumber is indexed from 1.
columnWidth, <*columnNumber*>	width. Note that columnNumber is indexed from 1. width is 1/256 of a character.

16. Excel Colors

One of the following:

aqua, automatic, black, black1, blue, blueGrey, blue1, brightGreen, brightGreen1, brown, coral, cornflowerBlue, darkBlue, darkGreen, darkRed, darkTeal, darkYellow, gold, green, grey25%, grey40%, grey50%, grey80%, indigo, lavender, lemonChiffon, lightBlue, lightCornflowerBlue, lightGreen, lightOrange, lightTurquoise, lightTurquoise1, lightYellow, lime, maroon, oliveGreen, orange, orchid, paleBlue, pink, pink1, plum, red, red1, rose, royalBlue, seaGreen, skyBlue, tan, turquoise, turquoise1, violet, white, white1, yellow, yellow1.

17. Database XML Properties

Property	Possible Values
driverClass	A JDBC driver class.
databaseUrl	A JDBC url.
databaseUserName	Database user.

databasePassword	User's password.
autoCommit	true or false. Default is true.
databaseHeaders	true or false (Used by IT variable processing). Default is false.
emptyColumn	Any string. Default is "".
sql	The sql to execute by updateDatabase().
sqlQuery	The query to execute by queryDatabase(), openDatabaseQuery() or IT variable processing.
:parameter	The value to substitute in the sql.

18. Database Functions

In addition to querying database tables using the IT variable, Total Access provides the following functions. Please see the csv2db.awk, db2excel.awk, excelDbExample.awk, excelDbExample2.awk and excelDbExample3.awk examples in the installation directory for more information:

- statementHandle = **queryDatabase**(queryProperties, outputArrays, [outputLines])

  Executes a SELECT query and returns the results in the sorted arrays outputArrays and optionally outputLines. outputArrays is an array of arrays and outputLines is an array of lines, with each entry indexed by row number (starting at 1). Columns in each line of the outputLines array are separated by the value of the IFS variable. Returns a handle used by closeDatabaseStatement().

- statementHandle = **updateDatabase**(inputProperties, [result])

  Executes non-query sql using the configuration specified in the input properties array, and returns any result in result[1]. Returns a handle used by closeDatabaseStatement().

- statementHandle = **openDatabaseQuery**(queryProperties)

  Opens a SELECT query using the configuration specified in the query properties array. Returns a handle used by getDatabaseRow() and closeDatabaseStatement().

- line = **getDatabaseRow**(statementHandle, [emptyColumnValue], [outColumns])

  Returns the next row from a SELECT query as a line. Columns in the line

are separated by the value of the IFS variable. Columns are also stored in the outColumns array (if specified). If emptyColumnValue is set, this value will be returned for NULL columns (otherwise empty string ("") is returned). A NUL string ("\0") is returned when no more rows exist. The statementHandle is the value returned by openDatabaseQuery().

- **closeDatabaseStatement**(statementHandle)

  Closes a database statement used by queryDatabase(), updateDatabase() or openDatabaseQuery(), freeing resources.

19. XML Functions

In addition to reading XML files using the IT variable, Total Access provides the following functions. Please see the xmlExample.awk example in the Total Access installation directory for more information:

- attrArray = **getXmlNodeAttrs**(node, attrArray)

  Returns a sorted array of the node's attributes.

- nameStr = **getXmlNodeName**(node)

  Returns the node's name.

- valueStr = **getXmlNodeValue**(node)

  Returns the node's value (or empty string ("") if the node doesn't have a value).

- isNul = **isXmlNulNode**(node)

  Returns 1 if the node is a NUL node; 0 otherwise.

- childNode = **getXmlNode**(node, xpathQuery, [normalize], [xmlHeader], [xmlIndent])

  Returns a child node based on the XPath query. A NUL node is returned if a node isn't associated with the query. If normalize is 1, the node is normalized (that is, combining text and CDATA sections). If xmlHeader is 1, the XML header is returned when the node is converted into an XML string. If xmlIndent is 1, the XML returned when the node is converted into a string is indented.

- nodeArray = **getXmlNodeArray**(node, xpathQuery, nodeArray, [normalize], [xmlHeader], [xmlIndent])

Returns a sorted node array based on the XPath query. Each node in the array is indexed with a consecutive integer (starting at 1). If normalize is 1, each node is normalized (that is, combining text and CDATA sections). If xmlHeader is 1, the XML header is returned when each node is converted into an XML string. If xmlIndent is 1, the XML returned when each node is converted into a string is indented.

- node = **createXmlNode**(xmlStr, xpathQuery, [normalize], [xmlHeader], [xmlIndent])

  Parses an XML string and returns a node based on the XPath query. A NUL node is returned if a node isn't associated with the query. If normalize is 1, the node is normalized (that is, combining text and CDATA sections). If xmlHeader is 1, the XML header is returned when the node is converted into an XML string. If xmlIndent is 1, the XML returned when the node is converted into a string is indented.

- nodeArray = **createXmlNodeArray**(xmlStr, xpathQuery, nodeArray, [normalize], [xmlHeader], [xmlIndent])

  Parses an XML string and returns a sorted node array based on the XPath query. Each node in the array is indexed with a consecutive integer (starting at 1). If normalize is 1, each node is normalized (that is, combining text and CDATA sections). If xmlHeader is 1, the XML header is returned when each node is converted into an XML string. If xmlIndent is 1, the XML returned when each node is converted into a string is indented.

- node = **readXmlNode**(fileName, xpathQuery, [normalize], [xmlHeader], [xmlIndent])

  Reads an XML file and returns a node based on the XPath query. A NUL node is returned if a node isn't associated with the query. If normalize is 1, the node is normalized (that is, combining text and CDATA sections). If xmlHeader is 1, the XML header is returned when the node is converted into an XML string. If xmlIndent is 1, the XML returned when the node is converted into a string is indented.

- nodeArray = **readXmlNodeArray**(fileName, xpathQuery, nodeArray, [normalize], [xmlHeader], [xmlIndent])

  Reads an XML file and returns a sorted node array based on the XPath query. Each node in the array is indexed with a consecutive integer (starting at 1). If normalize is 1, each node is normalized (that is, combining text and CDATA sections). If xmlHeader is 1, the XML header is returned when each node is converted into an XML string. If xmlIndent is 1, the XML returned when each node is converted into a string is

indented.

20. CSV Functions

In addition to reading CSV files using the IT variable, Total Access provides the following functions. Please see the csv2db.awk, csv2excel.awk and fixedWidth2csv.awk examples in the Total Access installation directory for more information:

- handle = **openCsv**(fileName, [delimiter])

  Opens an existing CSV file. If delimiter is not specified, it defaults to comma (“,”). Returns a handle for all subsequent operations.

- line = **getCsvLine**(handle, [emptyRowValue], [emptyColumnValue], [outFields], [toNumber])

  Returns the next line in a CSV file. Fields in the line are separated by the value of the IFS variable. Fields are also stored in the sorted outFields array (if specified), with each entry indexed with a consecutive integer (starting at 1). If toNumber is 1, Total Access attempts to convert the field into a number before storing it in the outFields array. If emptyRowValue is set, this value will be returned for empty rows (otherwise “null” is returned). If emptyColumnValue is set, this value will be returned for empty fields (otherwise empty string (“”) is returned). A NUL string (“\0”) is returned when no more rows exist. The handle is the value returned by openCsv().

- **closeCsv**(handle)

  Closes a CSV file. The handle is the value returned by openCsv().

- line = **parseCsvLine**(inputLine, [delimiter], [emptyRowValue], [emptyColumnValue], [outFields], [toNumber])

  Parses the CSV inputLine and returns a line. Fields in the line are separated by the value of the IFS variable. Fields are also stored in the sorted outFields array (if specified), with each entry indexed with a consecutive integer (starting at 1). If delimiter is specified, it is used as the delimiter for the CSV input; otherwise comma (“,”) is used. If emptyRowValue is set, this value will be returned for empty rows (otherwise “null” is returned). If emptyColumnValue is set, this value will be returned for empty fields (otherwise empty string (“”) is returned). If toNumber is 1, Total Access attempts to convert the field into a number before storing it in the outFields array.

- str = **unescapeCsv**(str)

  Returns an unescaped CSV string.

- str = **escapeCsv**(str)

  Returns an escaped CSV string.

21. Fixed Width Functions

Please see the fixedWidth2csv.awk and fixedWidth2excel.bat (or fixedWidth2excel.sh for Mac/Linux) examples in the Total Access installation directory:

- line = **parseFixedLine**(inputLine, widths, [outFields], [toNumber])

  Parses the fixed width input and returns a line. Fields in the line are separated by the value of the IFS variable. Fields are also stored in the sorted outFields array (if specified), with each entry indexed with a consecutive integer (starting at 1). The widths array specifies the width of each field (and white space). If toNumber is 1, Total Access attempts to convert the field into a number before storing it in the outFields array.