lccSQLDataExport Whitepaper

Table of Contents

Description

This document describes the technical details of the lccSQLDataExport program

About lccSQLDataExport

lccSQLDataExport was created to export data from SQL servers. This data could either be viewed or exported to a file. As the data is exported, it could be manipulated.

This need came about with the introduction of some new tools (HPSA, ODS, ADP) that allow institutions to download data from the state database system (HP3000) to local SQL servers.

The lccSQLDataExport application does not worry about the storage of data or the database engine for querying information, but, rather handles receiving the records from a query and manipulating as needed. One specific need that arose for this tool was that some columns in the HP3000 database are actually multiple fields in one. For example, a students name is stored in a column as DOE JOHN TIM. This tool will allow you to export the data and split that column into multiple columns as the data is read/exported. In this example, you can also designate the minimum and maximum number of columns to produce from splitting one column. This handles situations where information (like a name) may have only two names, while other records may contain a name with 5 parts, but, your requirements may need a specific numbers of columns for other programs/need, like [LAST], [FIRST], [MIDDLE]. By designating a minimum, the application will append column separators if the data is too few parts, and concatenate parts if it is too many parts.

Update 201202017
A new enhancement setting (queryStatementFillIn...) allow the SQL Queries to be modified by non-SQL users outside of the execution location.

Update 20200108
Tab delimited Logic Files now supported, see 'lcc:logicPath'. Adhoc user input now supported in the Tab delimited Logic Files, see 'lcc:userInput'.

Update 20200206
Added large download memory management. The program will by default, process every 10,000 records while downloading. You can modify this with key "lcc:flushmaxRecords".

Installation

This application is a command line application and requires no installation.

Note: as of Apr 20, 2020, we will only be updating the 32bit unless a need arises for the 64 bit version.

[update 11/14/2019] Along with the lccSQLDataExport.exe, there is now additional files to support Oracle:

oci.dll
ociw32.dll
oracle.dataaccess.dll
orannzsbb12.dll
oraociei12.dll
oraons.dll
oraops12.dll

This allows for connections to Oracle. Just place that file in the same folder as the lccSQLDataExport.exe.

To get help on the program, just use the syntax:

LCCSQLDataExport.exe ?

The application configures itself each time by the logic file you provide. By default, it will look for a logic file called:

LCCSQLDataExport-Logic.xml

but, you may supply any logic filename (to organize your exports).

Edit the Logic XML file to instruct lccSQLDataExport on how to run. Each time the program executes, it reads the logic file for how it should perform. You can make changes to the file at anytime. Those changes become effective immediately. The logic file should reside in the same location as the program, unless you specify a path.

An example logic fill is included in the installation package.

There are two fields that should be (highly recommended) encrypted. These are the SQLConnectionUserId and SQLConnectionPassword, if used. To produce encrypted data, run the application with the following syntax:

LCCSQLDataExport.exe "logicKey:[your Logic Key]" "encrypt:[your SQL UserId]"

-or-

LCCSQLDataExport.exe "logicKey:[your Logic Key]" "encrypt:[yourSQLPassword]"

If no User Id provided, will default to Integrated Security (i.e. no Id/Password).

The Logic Key is the same one used in your Logic File. This is used as the key to encrypt/decrypt the encrypted values. As long as you provide the same value when encrypting and in the logic file, the data can be read correctly.

Then place the encrypted value(s) into your logic file.

Command Line Parameters

These command line parameters can be provided to the program.

lccSQLDataExport.exe lcc:logicPath .\logic\adhoc-download-dataLinkStaging-remoteNoDB-logic.txt lcc:paramId lccParamDatabaseTarget lcc:paramValue ourTargetDB lcc:paramId lccParamSchemaTarget lcc:paramValue ourTargetSchema lcc:paramId lccParamTableTarget lcc:paramValue ourTargetTable lcc:paramId lccParamLinkServ lcc:paramValue ourLinkServ lcc:paramId lccParamDatabaseSource lcc:paramValue ourSourceDatabase lcc:paramId lccParamSchemaSource lcc:paramValue ourSourceSchema lcc:paramId lccParamTableSource lcc:paramValue ourSourceTable

logicFilename:

Where to pull Logic Settings from. This version uses a XML file.
Note: there is a colon between the key name "logicFilename" and the value, with no space. This is to support older versions of the program. All current/new keys use a space between the name and value.

You can use this or the lcc:logicPath (tab delimieted) option.

Syntax: logicFilename:[...path...]

Example: logicFilename:.\logic\ourLogic.txt

lcc:logicPath

Where to pull Logic Settings from. This version uses a tab delimited file.

You can use this or the logicFilename: (XML) option.

Syntax: lcc:logicPath *[tab]* [...path...]
Example: lcc:logicPath .\logic\ourLogic.txt

lcc:paramId

Creates an adhoc key id that can be used in key/value replacing within queries. More than one can be defined, with each lcc:paramValue value being paired with the last lcc:paramId defined.

Syntax: lcc:paramId *[tab]* [...Id...]
Example: lcc:paramId ourKeyId

lcc:paramValue

Pairs an adhoc value with the last key id created.

Syntax: lcc:paramValue *[tab]* [...Value...]
Example: lcc:paramValue ourValue

Example of using multiple adhoc key id/values:

lcc:paramId ourKeyId lcc:paramValue ourValue lcc:paramId ourKeyId2 lcc:paramValue ourValue2

Logic File Description/Syntax

A Logic File is a Tab delimited text file. Any lines not recognized as a valid Key/Value pair, will be ignore and can be used as remarks/other.

The Logic File uses the syntax.
Syntax: **[Key]** *[tab]* **[Value]** … *[tab]* **[Value]**
Example: lcc:key value

Any extra tabs in a line after the expected ones are considered remarks and will be ignored. This is a nice way to document specific Key settings (see Log Levels in the Logic File example(s) for reference). Also, if you place a tab before a line, that will essentially make it a remark and will be ignored, which makes using/not using logic without removing quicker.

Any line not starting with an expected key is ignored, which makes placing remarks/formatting the logic easy.

Logic File

If using a Tab Delimited Logic File, user adhoc input is supported. This allows a user to supply values on-the-fly (adhoc) when the process is ran. Any Key's value can be replaced with this flag.

For Tab Delimited Logic Files Only

To use user adhoc input, place "lcc:userInput" as the value, plus a space, and finally the prompt (question) to be shown to the user.

Syntax: lcc:userInput [space] SQL Query
Example: lcc:SQLQuery lcc:userInput SQL Query

This would result in the user receiving a prompt/question of "SQl Query", then whatever they supply will be the value for the "lcc:SQLQuery" Key.

lcc:logicFromFile

Will dynamically pull Logic from another file and insert into the current Logic File.

Syntax: lcc:logicFromFile *[tab]* [...Path...]
Example: lcc:logicFromFile .\logic\ourLogic.txt

lcc:recordsFoundHeartbeat

How often the program gives a heartbeat (status) on how many records downloaded. The default is 1000.

This must be a number.

Syntax: lcc:recordsFoundHeartbeat *[tab]* [#]
Example: lcc:recordsFoundHeartbeat 1000

lcc:flushMaxRecords

To avoid memory overload, the program will flush out/process downloaded records in sets. The default is 10000.

This must be a number.

Syntax: lcc:flushMaxRecords *[tab]* [#]
Example: lcc:flushMaxRecords 10000

lcc:exportMaxRecords

To avoid downloading too large of tables, set this key to limit how many rows per table to download. If set to '0', it will have no limit. The default value is '1000000' (1 million).

This must be a number.

Syntax: lcc:exportMaxRecords *[tab]* [1000000]
Example: lcc:exportMaxRecords 1000000

lcc:dataColumn (optional)

This setting is only a parent setting for other sub-settings. Only other XML settings are embedded, no values. You can specify one of these settings for each column defined in the dataColumns.

Note: if using a Tab delimited Logic File, start and end this section with:
lcc:dataColumn START
...
lcc:dataColumn END

The valid sub-settings available are:
dataColumnId
dataColumnChangeFormat
dataColumnSplitOn
dataColumnSplitMinimum
dataColumnSplitMaximumseparator

Syntax: lcc:dataColumn *[tab]* [START/END]
Example: lcc:dataColumn START

lcc:dataColumnId

A column name specified in the dataColumns setting. Any actions you specify in this section will be applied to this column.

Syntax: lcc:dataColumnId *[tab]* [...Id...]
Example: lcc:dataColumnId FullName

lcc:dataColumns

This is where you define how many columns will be returned by your SQLQuery and what names you wish to give them. The names do not need to match the actual names returned by the Query. For example, if your query is like SELECT sid, stu_name, stu_phone ...., this will return three columns of data. However, you can call these anything you want in this setting.

See lcc:dataColumnsAuto to set these automatically.

Syntax: lcc:dataColumns *[tab]* [Column Name,Column Name,...]
Example: lcc:dataColumns Id,FullName,Phone

lcc:dataColumnsAuto

This will auto build the Column Names from the query results (i.e. table schema for columns requested).

See lcc:dataColumns to set these manually.

Valid Values

- **YES**

Syntax: lcc:dataColumnsAuto *[tab]* [YES]
Example: lcc:dataColumnsAuto YES

lcc:dataColumnsAutoFilterOut

If key "lcc:dataColumnsAuto" used, this key can be provided to filter out any column names you want excluded from auto building, ex: "rowguid"

You can have one to many.

Syntax: lcc:dataColumnsAutoFilterOut *[tab]* [rowguid]
Example: lcc:dataColumnsAutoFilterOut rowguid

lcc:dataColumnsQualifyAll

By default, the program will only qualify column values (surround with double quotes), if the value contains a column separator or double quote. This key will qualify every column.

See key 'lcc:dataColumnsQualifyAllExcludeEmpty' for not qualifying empty columns.

Valid Values

- **YES**

Syntax: lcc:dataColumnsQualifyAll *[tab]* [YES]
Example: lcc:dataColumnsQualifyAll YES

lcc:dataColumnsQualifyAllExcludeEmpty

If key 'lcc:dataColumnsQualifyAll' used and a column is empty and this key is provided, this column will not be qualified (double quoted)

Valid Values

- **YES**

Syntax: lcc:dataColumnsQualifyAllExcludeEmpty *[tab]* [YES]
Example: lcc:dataColumnsQualifyAllExcludeEmpty YES

lcc:dataColumnsHeaders (optional)

This is where you define columns headers that will show as the first record in the exported data file. The names do not need to match the actual names returned by the Query. For example, you may instruct the program to split the FullName column into two columns, then designate LastName, FirstName column headers to show for the two columns created.

Syntax: lcc:dataColumnsHeaders *[tab]* [Id,Id,...]
Example: lcc:dataColumnsHeaders Id,LastName,FirstName,Phone

lcc:dataColumnsHeadersAuto

This will auto build the Column Headers from the query results (i.e. table schema for columns requested).

See lcc:dataColumns to set these manually.

Valid Values

- **YES**

Syntax: lcc:dataColumnsHeadersAuto *[tab]* [YES]
Example: lcc:dataColumnsHeadersAuto YES

lcc:dataColumnsHeadersAutoFilterOut

If key "lcc:dataColumnsHeadersAuto" used, this key can be provided to filter out any column names you want excluded from auto building, ex: "rowguid"

You can have one to many.

Syntax: lcc:dataColumnsHeaderAutoFilterOut *[tab]* [...Id...]
Example: lcc:dataColumnsHeadersAutoFilterOut rowguid

lcc:dataColumnChangeFormat (optional)

Change the value to another format.

Valid Formats:

- YYMMDD
- YYYYMM
- YYYYMM[tab]HH:MM:SS.MS
- YYYYMMDDHHMMSS
- YYYYMMDD
- YYYYMMDDHHMMSSMS
- YY
- MM/DD/YYYY
- HHMMSSMS
- HH:MM:SS
- HH:MM:SS.MS

Syntax: lcc:dataColumnChangeFormat *[tab]* [...Format...]

Example: lcc:dataColumnChangeFormat HH:MM:SS

lcc:dataColumnSplitAt (optional)

A value used to split a columns data. Each number designates at what position to split the column. You can use one or many numbers (separated by a comma).

Syntax: lcc:dataColumnSplitAt *[tab]* [#,#,...]
Example: lcc:dataColumnSplitAt 5,8

lcc:dataColumnSplitOn (optional)

A value used to split a columns data. Anything you enter in this setting will be used. If you want to separate by a [space], then enter an actual space. If you want a [tab], then type a tab. This field does not have to be a single character.

Syntax: lcc:dataColumnSplitOn *[tab]* [...character...]
Example: lcc:dataColumnSplitOn ,
Example #2: lcc:dataColumnSplitOn [tab]

lcc:dataColumnSplitMinimum (optional)

This setting will define the minimum number of times data in a column is split. For example, if you split 'David Mielcarek' on a space, which would result in two columns (since there is only one space), but, you set the minimum as '5', the application will add an additional 3 (three) blank columns to the output.

Syntax: lcc:dataColumnSplitMinimum *[tab]* [#]
Example: lcc:dataColumnSplitMinimum 5

lcc:dataColumnSplitMaximum (optional)

This setting will define the maximum number of times data in a column is split. For example, if you split 'David Arthur Mielcarek' on a space, which would result in three columns (since there are two spaces), but, you set the maximum as '2', the application will only split between 'David' and the rest, resulting in the second column having 'Arthur Mielcarek'.

Syntax: lcc:dataColumnSplitMaximum *[tab]* [#]
Example: lcc:dataColumnSplitMaximum 2

lcc:dataColumnSplitMaximumChop (optional)

This setting will force the data to split at all separators, but, will delete any columns that are created above the SQLDataColumnSplitMaximum value. For example, if you split 'David Arthur Mielcarek' on a space, which would result

in three columns (since there are two spaces), but, you set SQLDataColumnSplitMaximum as '2' and set this setting as 'YES', the data would split into David [separator] Arthur [separator] Mielcarek, but, would only save the David and Arthur columns.

Valid Values
- **YES**
- **NO**

Syntax: lcc:dataColumnSplitMaximumChop *[tab]* [YES/NO]
Example: lcc:dataColumnSplitMaximumChop YES

lcc:dataColumnsSeparator (optional)

A separator value used to separate the columns in the Export File. Example: ~

If using Tab Delimited Logic File, use "[TAB]" for the value if you want a Tab delimited output file.

lcc:dataColumnSwapColumns (optional)

A value used to swap columns. Each column swap will be a pair of column numbers, i.e. "1,3" (would swap column 1 with column 3). Up to 20 pairs can be designated. If you designate more than one pair, separate them with a ':' (colon), i.e. "1,3:2,4".

Syntax: lcc:dataColumnSwapColumns *[tab]* [#,#:,...]
Example: lcc:dataColumnSwapColumns 1,3 (swaps columns 1 & 3)
Example #2: lcc:dataColumnSwapColumns 2,5:1,7 (swaps columns 2 & 5, then 1 & 7)

lcc:dataColumnTrimLeftOn (optional)

This setting will trim a column on the characters you specify (separated by commas). This is done after any splitting.
Example: [space],# (this would trim any spaces and #'s from the left of the column. The [space] represents typing a space in your logic file, do not type the [space] shown here).
Note: if using an XML Logic format, XML does not like some special characters. You will need to use HTML type formatting for these, like the ampersand ('&'), you would use &, like: [space],&

Syntax: lcc:dataColumnTrimLeftOn *[tab]* [character,#]
Example: lcc:dataColumnTrimLeftOn :,3
Example: lcc:dataColumnTrimLeftOn [space],3

lcc:dataColumnTrimRightOn (optional)

This setting will trim a column on the characters you specify (separated by commas). This is done after any splitting.
Example: [space],# (this would trim any spaces and #'s from the right of the column. The [space] represents typing a space in your logic file, do not type the [space] shown here).
Note: XML does not like some special characters. You will need to use HTML type formatting for these, like the ampersand ('&'), you would use &, like: [space],&

Syntax: lcc:dataColumnTrimRightOn *[tab]* [character,#]
Example: lcc:dataColumnTrimRightOn :,3
Example: lcc:dataColumnTrimRightOn [space],3

lcc:dataExportFilename (optional)

The file to be created, containing the data from the SQL query, and modified as desired.

See **Date/Time Flags** section for auto embedding date/times into the path/filename. Also see 'lcc:queryStatementFillIn…' for auto changing part of the value.

This key can be used more than one if you supply the key 'lcc:SQLQuery' more than once. See the 'lcc:SQLQuery' key definition for more information.

Syntax: lcc:dataExportFilename *[tab]* [...path...]
Example: lcc:dataExportFilename .\ourDataFile.txt
Example #2: lcc:dataExportFilename \\server\share$\folder\ourDataFile.txt

lcc:dataRecordsMaxPerFile (optional)

This is where you define if the exported records should only contain a maximum set of records per file. If assigned, this will create additional files for each set of records, using the DataExportFilename, with appended "-#" on each file created beyond the first set. Using the example and an export filename of 'records.txt', with 5300 records, you would end up with three files 'records.txt', 'records-2.txt' and 'records-3.txt'. If not extension is found on the filename (i.e. no period in the filename), the "-#" will be placed on the end of the filename.

Syntax: lcc:dataRecordsMaxPerFile *[tab]* [#]
Example: lcc:dataRecordsMaxPerFile 2000

lcc:valuesFlagReplaceCRLF (optional)

Will replace any Carriage Return (CR) and/or Line Feed (LF) with this value.

This is useful if you have columns that include multiple lines (like descriptions).

If you provide a white-space character, like space, all CR/LF will be replaced with that character.

Flags available:

- **[lccFlag:CRLF]**

Syntax: lcc:valuesFlagReplaceCRLF *[tab]* [...character...]
Example: lcc:valuesFlagReplaceCRLF [space]

lcc:valuesFlagReplaceTab (optional)

Will replace any Tab with this value.

This is useful if you have columns that include tabs and you are exportingn to a tab delimited file.

If you provide a white-space character, like space, all Tabs will be replaced with that character.

Flags available:

- **[lccFlag:CRLF]**

Syntax: lcc:valuesFlagReplaceTab *[tab]* [...character...]
Example: lcc:valuesFlagReplaceTab [space]

lcc:exportColumnDelete (optional)

A column number that specifies which column in the finished record, will be deleted, i.e. not saved. The first column would be '**1**'. If you have instructed a column to be split, the new number of columns should be considered when assigning this number. For example, if the program receives a column with a name like "DOE JOHN SMITH" and you have instructed it to be split on [spaces], the exported record would have two more columns, i.e. the single column DOE JOHN SMITH, would become the three columns DOE;JOHN;SMITH.

Can define multiple by supplying this setting more than once.

Syntax: lcc:exportColumnDelete *[tab]* [#]
Example: lcc:exportColumnDelete 2

lcc:exportReplaceSource (optional)

If this is supplied, along with the ExportReplaceTarget, any matches found in an exported column that match this setting, will be replaced with the ExportReplaceTarget.

Can define multiple by supplying this and the ExportReplaceTarget setting more than once.

Syntax: lcc:exportReplaceSource *[tab]* [...value...]
Example: lcc:exportReplaceSource file

lcc:exportReplaceTarget (optional)

If this is supplied, along with the ExportReplaceSource, any matches found in an exported column that match will be replaced with this setting.

Can define multiple by supplying this and the ExportReplaceSource setting more than once.

Syntax: lcc:exportReplaceTarget *[tab]* [...value...]
Example: lcc:exportReplaceTarget filename

lcc:exportColumnPosition (optional)

A column number that specifies which column in the finished record, will be manipulated by any ExportColumn actions. The first column would be '**1**'. If you have instructed a column to be split, the new number of columns should be considered when assigning this number. For example, if the program receives a column with a name like "DOE JOHN SMITH" and you have instructed it to be split on [spaces], the exported record would have two more columns, i.e. the single column DOE JOHN SMITH, would become the three columns DOE;JOHN;SMITH.

Syntax: lcc:exportColumnPosition *[tab]* [#]
Example: lcc:exportColumnPosition 2

lcc:exportColumnMinLength (optional)

Must define lcc:exportColumnPosition first.
A column that has less characters than the number defined here, will be right padded with spaces to make its length equal the minimum length. For example, if a column ended up with 'SMITH' and the minimum was set at '10', it would result in a column with 'SMITH ' (5 spaces on the end).

Syntax: lcc:exportColumnMinLength *[tab]* [#]
Example: lcc:exportColumnMinLength 10

lcc:exportColumnMaxLength (optional)

Must define lcc:exportColumnPosition first.
A column that has more characters than the number defined here, will be right truncated. For example, if a column ended up with 'SMITH' and the maximum was set at '2', it would result in a column with 'SM'.

Syntax: lcc:exportColumnMaxLength *[tab]* [#]
Example: lcc:exportColumnMaxLength 2

lcc:exportColumnCapitalize (optional)

Must define lcc:exportColumnPosition first.
Will capitalize the value if the value of this key is set to 'YES'. The first character will be set to Upper Case, and all other characters to Lower Case.

Valid Values:

- **YES**
- **NO**

Syntax: lcc:exportColumnCapitalize *[tab]* [YES]
Example: lcc:exportColumnCapitalize YES

lcc:exportColumnPadLeft (optional)

Must define lcc:exportColumnPosition first.
A character used in padding (see ExportColumnMinLength) the value of a column. Only the first character is used, if more than one supplied. The padding will happen on the left.

Syntax: lcc:exportColumnPadLeft *[tab]* [...character...]
Example: lcc:exportColumnPadLeft 0

lcc:exportColumnPadRight (optional)

Must define lcc:exportColumnPosition first.
A character used in padding (see ExportColumnMinLength) the value of a column. Only the first character is used, if more than one supplied. The padding will happen on the right. If ExportColumnPadLeft is supplied, this key will be ignored.

Syntax: lcc:exportColumnPadRight *[tab]* [...character...]

Example: lcc:exportColumnPadRight 0

lcc:logicEncrypted

Instructs the program whether your senstiive fields are encrypted. When you first install the program, your values will probably be plain text, as you need to have the program running to use the Encrypt tool. However, after install, it is highly recommended that you change this to YES and encrypt your values. Only the sensitive fields will be decrypted, the rest can be plain text. See field descriptions in this section to see if they are noted for encryption.

Valid Values:

- **YES**

Syntax: lcc:logicEncrypted *[tab]* [YES]
Example: lcc:logicEncrypted YES

lcc:logicKey

The key used to encrypt/decrypt security values. This key can be any phrase. It is used to change how the encrypt/decrypt formula works. This means if you encrypt your values using 'cat' as your logic key, and later change the logic key to 'dog', you need to re-encrypt the values, or they will not decrypt successfully.

Syntax: lcc:logicKey *[tab]* [...key...]
Example: lcc:logicKey some cool phrase

lcc:queryStatementFillIn (optional)

This setting is only a parent setting for other sub-settings. Only other XML settings are embedded, no values. You can specify one of these settings for each column defined in the queryStatementFillIn. The queryStatementFillIn... settings allow you to have queries that can be modified by others. For example, you may want to allow non-SQL users to modify the Quarter Code in a query.

Note: if using a Tab delimited Logic File, start and end this section with:
lcc:queryStatementFillIn START
...
lcc:queryStatementFillIn END

The valid sub-settings available are:
queryStatementFillInId
queryStatementFillInPath

Syntax: lcc:queryStatementFillIn *[tab]* [START/END]
Example: lcc:queryStatementFillIn START

lcc:queryStatementFillInId

An Id specified in the queryStatementFillIn setting. This is a string of characters that will identify where to Fill-In information inside the "lcc:SQLQuery" or "lcc:dataExportFilename" . Within the query, you would designate this replacement by putting this Id surrounded by "[~" and "~]"

Example how Id is entered in SQLQuery: [~replaceMe1~]

Syntax: lcc:queryStatementFillInId *[tab]* [...Id...]
Example: lcc:queryStatementFillInId replaceMe1

lcc:queryStatementFillInPath

A Path specified in the queryStatementFillIn setting. This is a path to a file containing the data that will replace the Id. Any (all lines) data will be inserted into the "lcc:SQLQuery" or "lcc:dataExportFilename" values.

Syntax: lcc:queryStatementFillInPath *[tab]* [...path...]
Example: lcc:queryStatementFillInPath \\server\share$\folder\file.txt

lcc:runPostCommand

Instructs the program to run a command after processing is finished. If you want multiple commands, add multiple lines of this key. The commands will run asynchronously, i.e. they will not wait for each other.

Syntax: lcc:runPostCommnad *[tab]* [...command...]
Example: lcc:runPostCommand copy ourFile.txt \\server\share$\folder /Y
Example #2 (multiple):
lcc:runPostCommand copy ourFile.txt \\server\share$\folder /Y
lcc:runPostCommand runOurFilter.exe param1

lcc:sourceRemoveConsecutiveChar (optional)

Will remove any consecutive characters matching the first character of this setting. For example, if you supply a [space], any imported records will not have more than one space consecutively.

Can define multiple by supplying this setting more than once.

This example would remove any consecutive '2' characters. If your value had 'Phone 25224225', it would end up with '252425'.

Syntax: lcc:sourceRemoveConsecutiveChar *[tab]* [...character...]
Example: lcc:sourceRemoveConsecutiveChar 2

lcc:SQLConnectionWorkstationId

What computer name will be provided to the SQLConnectionDataSource (SQL server).

Syntax: lcc:SQLConnectionWorkstationId *[tab]* [...Id...]
Example: lcc:SQLConnectionWorkstationId LCCInHouseProgram

lcc:SQLConnectionTimeout

Change the default connection timeout, in seconds.

Syntax: lcc:SQLConnectionTimeout *[tab]* [#]
Example: lcc:SQLConnectionTimeout 600

lcc:SQLConnectionCommandTimeout

Change the default connection command timeout, in seconds.

Syntax: lcc:SQLConnectionCommandTimeout *[tab]* [#]
Example: lcc:SQLConnectionCommandTimeout 600

lcc:SQLConnectionPacketSize

What packet size will be used when communicating with the SQLConnectionDataSource (SQL server)..

Syntax: lcc:SQLConnectionPacketSize *[tab]* [#]
Example: lcc:SQLConnectionPacketSize 4096

lcc:SQLConnectionUserId

What LDAP account will be used to communicate with the SQLConnectionDataSource (SQL Server).
* This value should be encrypted using the Encrypt tool.

If not provided, will default to Integrated Security (i.e. no Id/Password).

Syntax: lcc:SQLConnectionUserId *[tab]* [...Id...]
Example: lcc:SQLConnectionuserId ourSQLAccount

lcc:SQLConnectionPassword

What password will be used in conjunction with the SQLConnectionUsserId.
* This value should be encrypted using the Encrypt tool.

Syntax: lcc:SQLConnectionPassword *[tab]* [...password...]
Example: lcc:SQLConnectionPassword *****

lcc:SQLConnectionDataSource

What FQDN or IP will be used for SQL actions.

Syntax: lcc:SQLConnectionDataSource *[tab]* [...server...]
Example: lcc:SQLConnectionDataSource ourSQLServer

lcc:SQLConnectionDataPort

What Port will be used for the SQL connection.

Only valid if using the Key 'lcc:SQLConnectionPlatform' with the value of 'Oracle'.

Syntax: lcc:SQLConnectionDataPort *[tab]* [#]
Example: lcc:SQLConnectionDataPort 1234

lcc:SQLConnectionInitialCatalog

What database is being used on the SQLConnectionDataSource (SQL server).

Syntax: lcc:SQLConnectionInitialCatalog *[tab]* [...database...]
Example: lcc:SQLConnectionInitialCatalog ourDatabase

lcc:SQLConnectionPlatform

What Platform is being connected to. If none provided, will default to SQL.

Note: when using Oracle, do not end your SQL command with a semicolon.

Valid Values

- (none), i.e. default to SQL
- Oracle

Syntax: lcc:SQLConnectionPlatform *[tab]* [Oracle]
Example: lcc:SQLConnectionPlatform Oracle

lcc:SQLQuery

This is the SQL query statement that will be passed to the SQL server. Its results will be returned and parsed through this application.

See **Date/Time Flags** section for auto embedding date/times into the Query. Also see 'lcc:queryStatementFillIn…' for auto changing part of the value.

A sub-key "[lccQueryFromFile:...]" can also be added. This key will pull query commands from a file and embed into the query where this key is placed. You can provide multiple of this key in the same query. For each file you want to embed, use this key and provide the file path after the "[lccQueryFromFile:" portion and enclose with the ending right bracket.

Multiple of these keys can be provided. Each one will be considered a full run, i.e. it will open the SQL connection, run the query, do all column/export files, then close the SQL connection for each query. If you use the key ' lcc:dataExportFilename', you can supply one of more of those to use for the quer(ies). If more than one, of this key and the 'lcc:dataExportFilename' key, each will be matched up with the other in order. If there are not enough of the 'lcc:dataExportFilename' keys, the last one defined will be used. For example, if you supply 5 queries, but only 2 files, query 1 would get file 1, while queries 2-5 will get file 2.

The following flags can be placed into the query. The program will auto change the query when ran:

- **[~WAYRQQuartersPrevious2Year~]** : will insert the four (4) quarter codes for the year that falls 2 years in the past. The output will end up like: B781,B782,B783,B784
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WAYRQQuartersPrevious2Year~]);
  If the current year is 2019, it would produce quarter codes for 2017 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (B781,B782,B783,B784);
- **[~WAYRQQuartersPreviousYear~]** : will insert the four (4) quarter codes for the year that falls 1 year in the past. The output will end up like: B891,B892,B893,B894
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WAYRQQuartersPreviousYear~]);
  If the current year is 2019, it would produce quarter codes for 2018 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (B781,B782,B783,B784);

- **[~WAYRQQuartersCurrentYear~]** : will insert the four (4) quarter codes for the year that falls in the current year. The output will end up like: B901,B902,B903,B904
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WAYRQQuartersCurrentYear~]);
  If the current year is 2019, it would produce quarter codes for 2019 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (B901,B902,B903,B904);
- **[~WAYRQQuartersNextYear~]** : will insert the four (4) quarter codes for the year that falls in the next year. The output will end up like: C011,C012,C013,C014
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WAYRQQuartersNextYear~]);
  If the current year is 2019, it would produce quarter codes for 2020 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (C011,C012,C013,C014);
- **[~WATermsPrevious2Year~]** : will insert the four (4) term codes for the year that falls 2 years in the past. The output will end up like: 2171,2173,2175,2177
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsPrevious2Year~]);
  If the current year is 2019, it would produce term codes for 2020 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2171,2173,2175,2177);
- **[~WATermsPreviousYear~]** : will insert the four (4) term codes for the year that falls 1 year in the past. The output will end up like: 2181,2183,2185,2187
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsPreviousYear~]);
  If the current year is 2019, it would produce term codes for 2019 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2181,2183,2185,2187);
- **[~WATermsCurrentYear~]** : will insert the four (4) term codes for the current year. The output will end up like: 2191,2193,2195,2197
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsCurrentYear~]);
  If the current year is 2019, it would produce term codes for 2019 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2191,2193,2195,2197);
- **[~WATermsNextYear~]** : will insert the four (4) term codes for the next year. The output will end up like: 2201,2203,2205,2207
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsNextYear~]);
  If the current year is 2019, it would produce term codes for 2020 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2201,2203,2205,2207);
- **[~WATermsPrevious4Term~]** : will insert a term code for 4 terms in the past. The output will end up like: 2191
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsPrevious4Term~]);
  If the current year is 2020 and Term 1, it would produce term codes for 2019, Term 1 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2191);
- **[~WATermsPrevious3Term~]** : will insert a term code for 3 terms in the past. The output will end up like: 2193

Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsPrevious3Term~]);
If the current year is 2020 and Term 1, it would produce term codes for 2019, Term 3 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2193);

- **[~WATermsPrevious2Term~]** : will insert a term code for 2 terms in the past. The output will end up like: 2195
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsPrevious2Term~]);
  If the current year is 2020 and Term 1, it would produce term codes for 2019, Term 5 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2195);
- **[~WATermsPreviousTerm~]** : will insert a term code for 1 term in the past. The output will end up like: 2197
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsPreviousTerm~]);
  If the current year is 2020 and Term 1, it would produce term codes for 2019, Term 7 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2197);
- **[~WATermsCurrentTerm~]** : will insert a term code for the current term. The output will end up like: 2201
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsCurrentTerm~]);
  If the current year is 2020 and Term 1, it would produce term codes for 2020, Term 1 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2201);
- **[~WATermsNextTerm~]** : will insert a term code for 1 term in the future. The output will end up like: 2203
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsNextTerm~]);
  If the current year is 2020 and Term 1, it would produce term codes for 2020, Term 3 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2203);
- **[~WATermsNext2Term~]** : will insert a term code for 2 terms in the future. The output will end up like: 2205
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsNext2Term~]);
  If the current year is 2020 and Term 1, it would produce term codes for 2020, Term 5 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2205);
- **[~WATermsNext3Term~]** : will insert a term code for 3 terms in the future. The output will end up like: 2207
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsNext3Term~]);
  If the current year is 2020 and Term 1, it would produce term codes for 2020, Term 7 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2207);
- **[~WATermsNext4Term~]** : will insert a term code for 4 terms in the future. The output will end up like: 2211
  Example: SELECT * FROM ourTable WHERE YRQ IN ([~WATermsNext4Term~]);
  If the current year is 2020 and Term 1, it would produce term codes for 2021, Term 1 and the query would end up like: SELECT * FROM ourTable WHERE YRQ IN (2211);

Example Sub-Key:
[lccQueryFromFile:ourQuery.txt]

Example: SELECT RTRIM(SM.STU_D.SID),RTRIM(SM.STU_D.STU_NAME),RTRIM(SM.CLASS_D.COURSE_TITLE),RIGHT(RTRIM(SM.STU_CLASS_D.STU_QTR),4),RTRIM(SM.CLASS_D.INSTR_NAME),RTRIM(SM.STU_CLASS_D.GR),RTRIM(SM.STU_CLASS_D.ADD_DATE) FROM SM.STU_D LEFT JOIN SM.STU_CLASS_D ON SM.STU_D.SID = LEFT(SM.STU_CLASS_D.STU_QTR,9) LEFT JOIN SM.CLASS_D ON SM.STU_CLASS_D.CLASS_ID = SM.CLASS_D.CLASS_ID WHERE RIGHT(RTRIM(SM.STU_CLASS_D.STU_QTR),4) = 'A901' ORDER BY SM.STU_D.SID;

Example, showing embedded query from file: [lccQueryFromFile:ourQuery.txt]

Syntax: lcc:SQLQuery *[tab]* [...query...]
Example: lcc:SQLQuery SELET * FROM ourTable;
Example #2: lcc:SQLQuery [lccQueryFromFile:.\queries\ourQuery.txt]

lcc:SQLQueryFromFileFromList

If you are using the Sub-Key '[lccQueryFromFile:...]' in the key 'lcc:SQLQuery to define a query from a file, you can use this key to run that query multiple times. This will allow you to define a file that contains a record (line) for each time it is to be run.

This comes in handy if you want to run a query different ways, like against different tables. With these records, you can define what 'flag' is in the query file to replace, what to replace it with, and what to prepend to the output filename so that each run of the query is in its own file.

Each line in the file should have the syntax:
[flag to replace] *[tab]* [replace with] *[tab]* [prepend to filename]
Examples:

tableId	Table1	diffQuery-table1-
tableId	Table2	diffQuery-table2-
tableId	Table2	diffQuery-table2-

For example, if you provided the line:
tableId Table1 diffQuery-table1-
The program will load the query file, replace all occurrences of 'tableId' with 'Table1' and save the results to the regular output file, but, with 'diffQuery-table1-' prepended to the filename.

Syntax: lcc:SQLQueryFromFileFromList *[tab]* [...path...]

Example: lcc:SQLQueryFromFileFromList \\server\share$\folder\ourQuery-differentTables.txt

Logic File Templates

A Logic File is a Tab delimited text fi

The logic file templates are as follows (explanations of each value follow the template):

There are two formats available; XML and Tab delimited. We recommend using Tab delimited.

XML

```
<?xml version="1.0" encoding="us-ascii" standalone="yes"?>
<lcc:lccSQLDataExport xmlns:lcc="lcc.ctc.edu" xmlns="lcc.ctc.edu">
<lcc:dataColumns></lcc:dataColumns>
<lcc:dataColumnsHeaders></lcc:dataColumnsHeaders>
<lcc:dataColumnsSeparator></lcc:dataColumnsSeparator>
<lcc:dataColumn>
        <lcc:dataColumnId></lcc:dataColumnId>
        <lcc:dataColumnSplitAt></lcc:dataColumnSplitAt>
        <lcc:dataColumnTrimLeftOn></lcc:dataColumnTrimLeftOn>
        <lcc:dataColumnTrimRightOn></lcc:dataColumnTrimRightOn>
        <lcc:dataColumnSplitMinimum></lcc:dataColumnSplitMinimum>
        <lcc:dataColumnSplitMaximum></lcc:dataColumnSplitMaximum>
</lcc:dataColumn>
<lcc:dataColumn>
        <lcc:dataColumnId></lcc:dataColumnId>
        <lcc:dataColumnSplitOn></lcc:dataColumnSplitOn>
        <lcc:dataColumnSplitMinimum></lcc:dataColumnSplitMinimum>
        <lcc:dataColumnSplitMaximum></lcc:dataColumnSplitMaximum>
</lcc:dataColumn>
<lcc:dataRecordsMaxPerFile>NUMBER OF RECORDS PER FILE</lcc:dataRecordsMaxPerFile>
<lcc:debugShowXML></lcc:debugShowXML>
```

```
<lcc:exportColumn>
        <lcc:exportColumnPosition></lcc:exportColumnPosition>
        <lcc:exportColumnMinLength></lcc:exportColumnMinLength>
        <lcc:exportColumnMaxLength></lcc:exportColumnMaxLength>
</lcc:exportColumn>
<lcc:logicEncrypted></lcc:logicEncrypted>
<lcc:logicKey></lcc:logicKey>
<lcc:SQLConnectionWorkstationId></lcc:SQLConnectionWorkstationId>
<lcc:SQLConnectionTimeout></lcc:SQLConnectionTimeout>
<lcc:SQLConnectionCommandTimeout></lcc:SQLConnectionCommandTimeout>
<lcc:SQLConnectionPacketSize></lcc:SQLConnectionPacketSize>
<lcc:SQLConnectionUserId></lcc:SQLConnectionUserId>
<lcc:SQLConnectionPassword></lcc:SQLConnectionPassword>
<lcc:SQLConnectionDataSource></lcc:SQLConnectionDataSource>
<lcc:SQLConnectionInitialCatalog></lcc:SQLConnectionInitialCatalog>
<lcc:queryStatementFillIn>
        <lcc:queryStatementFillInId></lcc:queryStatementFillInId>
        <lcc:queryStatementFillInPath></lcc:queryStatementFillInPath>
</lcc:queryStatementFillIn>
<lcc:SQLQuery></lcc:SQLQuery>
</lcc: lccSQLDataExport >
```

Tab Delimited

```
lcc:dataColumnsAuto          YES
lcc:dataColumnsHeadersAuto       YES
lcc:dataColumnsSeparator   [TAB]
lcc:dataRecordsMaxPerFile  NUMBER OF RECORDS PER FILE
lcc:logicEncrypted   YES
lcc:logicKey   our secret key
lcc:SQLConnectionWorkstationId   ...
lcc:SQLConnectionTimeout  ...
lcc:SQLConnectionCommandTimeout       ...
lcc:SQLConnectionPacketSize        ...
lcc:SQLConnectionUserId    ...
lcc:SQLConnectionPassword          ...
```

```
lcc:SQLConnectionDataSource       ...
lcc:SQLConnectionInitialCatalog   ...
lcc:queryStatementFillIn      START
lcc:queryStatementFillInId    ...
lcc:queryStatementFillInPath          ...
lcc:queryStatementFillIn      END
lcc:SQLQuery ...
```

Date/Time Flags

Certain key Values can have date/time flags embedded that instructs the program to place the current date/time in that location. For example, in the filename, or query.

The following flags are valid:

- [lccFlag:YYYYMM]
- [lccFlag:YYYYMM[tab]HH:MM:SS:MS]
- [lccFlag:YYYYMMDDHHMMSS]
- [lccFlag:YYYYMMDD]
- [lccFlag:YYYYMMDDHHMMSSMS]
- [lccFlag:HHMMSSMS]

Example: If you wanted the current YYYYMMDD in the filename, you could use:

```
<lcc:dataExportFilename>E:\folders\filename-[lccFlag:YYYYMMDD].txt</lcc:dataExportFilename>
```

This would end up with a file (on 4/13/2017) as: E:\folders\filename-20170413.txt

Execution

The lccSQLDataExport application is a 'command line' application. This allows use in batch files, task schedulers, etc.. Since the application can change how it works, using different logic files, you can have it run multiples times, with each occurence producing different data from different SQL servers.

Examples syntax

Running Program With XML Logic File

LCCSQLDataExport.exe "logicFilename:LCCSQLDataExport-studentInformation.xml"

Running Program With Tab Delimited Logic File

LCCSQLDataExport.exe lcc:logicPath "LCCSQLDataExport-studentInformation-logic.txt"

Running Program With Tab Delimited Logic File And Parameter Ids/Values

lccSQLDataExport.exe lcc:logicPath .\logic\adhoc-download-dataLinkStaging-remoteNoDB-logic.txt lcc:paramId lccParamDatabaseTarget lcc:paramValue ourTargetDB lcc:paramId lccParamSchemaTarget lcc:paramValue ourTargetSchema lcc:paramId lccParamTableTarget lcc:paramValue ourTargetTable lcc:paramId lccParamLinkServ lcc:paramValue ourLinkServ lcc:paramId lccParamDatabaseSource lcc:paramValue ourSourceDatabase lcc:paramId lccParamSchemaSource lcc:paramValue ourSourceSchema lcc:paramId lccParamTableSource lcc:paramValue ourSourceTable

Encrypting Data

LCCSQLDataExport.exe "logicKey:some cool phrase" "encrypt:ourSQLUserId "

Logic File Examples

Tab Delimited

```
lcc:dataColumnsAuto	YES
lcc:dataColumnsHeadersAuto	YES
lcc:dataColumnsSeparator	[TAB]
lcc:logicEncrypted	NO
lcc:logicKey	...
lcc:SQLConnectionWorkstationId	lccInHouseProgram
lcc:SQLConnectionPacketSize	4096
lcc:SQLConnectionUserId	ourId
```

lcc:SQLConnectionPassword ourPassword
lcc:SQLConnectionDataSource 123.123.123.123
lcc:SQLConnectionInitialCatalog ourDatabase
lcc:SQLQuery SELECT * FROM ourSchema.ourTable

XMLs

Queries a table with 3 columns.

```
<?xml version="1.0" encoding="us-ascii" standalone="yes"?>
<lcc:lccSQLDataExport xmlns:lcc="lcc.ctc.edu" xmlns="lcc.ctc.edu">
<lcc:debugMode>NO</lcc:debugMode>
<lcc:debugShowLogic>NO</lcc:debugShowLogic>
<lcc:dataColumns>OWNER,TABLE_NAME,OBJECT_TYPE</lcc:dataColumns>
<lcc:dataColumnsHeaders>OWNER,TABLE_NAME,OBJECT_TYPE</lcc:dataColumnsHeaders>
<lcc:dataColumnsSeparator>    </lcc:dataColumnsSeparator>
<lcc:dataExportFilename>ctcLinkQuery-PDB130-test-export.txt</lcc:dataExportFilename>
<lcc:valuesFlagReplaceCRLF> </lcc:valuesFlagReplaceCRLF>
<lcc:valuesFlagReplaceTab> </lcc:valuesFlagReplaceTab>
<lcc:debugShowGarble>NO</lcc:debugShowGarble>
<lcc:debugShowXML>YES</lcc:debugShowXML>
<lcc:SQLConnectionWorkstationId>ourWorkstation</lcc:SQLConnectionWorkstationId>
<lcc:SQLConnectionPacketSize>4096</lcc:SQLConnectionPacketSize>
<lcc:SQLConnectionUserId>ourUser</lcc:SQLConnectionUserId>
<lcc:SQLConnectionPassword>ourPassword</lcc:SQLConnectionPassword>
<lcc:SQLConnectionDataSource>123.123.123.123</lcc:SQLConnectionDataSource>
<lcc:SQLConnectionDataPort>123</lcc:SQLConnectionDataPort>
<lcc:SQLConnectionInitialCatalog>ourDatabase</lcc:SQLConnectionInitialCatalog>
<lcc:SQLQuery> SELECT OWNER,TABLE_NAME,OBJECT_TYPE FROM ourSchema.ourTable WHERE ROWNUM<3 ORDER BY
TABLE_NAME </lcc:SQLQuery>
</lcc:lccSQLDataExport>
```

Queries a table with all columns, auto building columns used and column headers

```
<?xml version="1.0" encoding="us-ascii" standalone="yes"?>
<lcc:lccSQLDataExport xmlns:lcc="lcc.ctc.edu" xmlns="lcc.ctc.edu">
<lcc:debugMode>NO</lcc:debugMode>
```

```
<lcc:debugShowLogic>NO</lcc:debugShowLogic>
<lcc:dataColumnsAuto>YES</lcc:dataColumnsAuto>
<lcc:dataColumnsHeadersAuto>YES</lcc:dataColumnsHeadersAuto>
<lcc:dataColumnsSeparator>    </lcc:dataColumnsSeparator>
<lcc:dataExportFilename>ctcLinkQuery-PDB130-test-export.txt</lcc:dataExportFilename>
<lcc:valuesFlagReplaceCRLF> </lcc:valuesFlagReplaceCRLF>
<lcc:valuesFlagReplaceTab> </lcc:valuesFlagReplaceTab>
<lcc:debugShowGarble>NO</lcc:debugShowGarble>
<lcc:debugShowXML>YES</lcc:debugShowXML>
<lcc:SQLConnectionWorkstationId>ourWorkstation</lcc:SQLConnectionWorkstationId>
<lcc:SQLConnectionPacketSize>4096</lcc:SQLConnectionPacketSize>
<lcc:SQLConnectionUserId>ourUser</lcc:SQLConnectionUserId>
<lcc:SQLConnectionPassword>ourPassword</lcc:SQLConnectionPassword>
<lcc:SQLConnectionDataSource>123.123.123.123</lcc:SQLConnectionDataSource>
<lcc:SQLConnectionDataPort>123</lcc:SQLConnectionDataPort>
<lcc:SQLConnectionInitialCatalog>ourDatabase</lcc:SQLConnectionInitialCatalog>
<lcc:SQLQuery> SELECT OWNER,TABLE_NAME,OBJECT_TYPE FROM ourSchema.ourTable WHERE ROWNUM<3 ORDER BY
TABLE_NAME </lcc:SQLQuery>
</lcc:lccSQLDataExport>
```

Queries a table with all columns, auto building columns used and column headers, with SQL commands coming from a file, on an Oracle system

```
<?xml version="1.0" encoding="us-ascii" standalone="yes"?>
<lcc:lccSQLDataExport xmlns:lcc="lcc.ctc.edu" xmlns="lcc.ctc.edu">
<lcc:debugMode>NO</lcc:debugMode>
<lcc:debugShowLogic>NO</lcc:debugShowLogic>
<lcc:dataColumnsAuto>YES</lcc:dataColumnsAuto>
<lcc:dataColumnsHeadersAuto>YES</lcc:dataColumnsHeadersAuto>
<lcc:dataColumnsSeparator>    </lcc:dataColumnsSeparator>
<lcc:dataExportFilename>ctcLinkQuery-PDB130-test-export.txt</lcc:dataExportFilename>
<lcc:valuesFlagReplaceCRLF> </lcc:valuesFlagReplaceCRLF>
<lcc:valuesFlagReplaceTab> </lcc:valuesFlagReplaceTab>
<lcc:debugShowGarble>NO</lcc:debugShowGarble>
<lcc:debugShowXML>YES</lcc:debugShowXML>
<lcc:SQLConnectionWorkstationId>ourWorkstation</lcc:SQLConnectionWorkstationId>
```

```
<lcc:SQLConnectionPacketSize>4096</lcc:SQLConnectionPacketSize>
<lcc:SQLConnectionUserId>ourUser</lcc:SQLConnectionUserId>
<lcc:SQLConnectionPassword>ourPassword</lcc:SQLConnectionPassword>
<lcc:SQLConnectionDataSource>123.123.123.123</lcc:SQLConnectionDataSource>
<lcc:SQLConnectionDataPort>123</lcc:SQLConnectionDataPort>
<lcc:SQLConnectionInitialCatalog>ourDatabase</lcc:SQLConnectionInitialCatalog>
<lcc:SQLConnectionPlatform>Oracle</lcc:SQLConnectionPlatform>
<lcc:SQLQuery>[lccQueryFromFile:ourSQLQuery-sql.txt]</lcc:SQLQuery>
</lcc:lccSQLDataExport>
```

Tab Delimited

Queries a table with 3 columns.

```
lcc:debugMode     NO
lcc:debugShowLogic      NO
lcc:dataColumns   OWNER,TABLE_NAME,OBJECT_TYPE
lcc:dataColumnsHeaders  OWNER,TABLE_NAME,OBJECT_TYPE
lcc:dataColumnsSeparator      [tab]
lcc:dataExportFilename  ctcLinkQuery-PDB130-test-export.txt
lcc:valuesFlagReplaceCRLF     [space]
lcc:valuesFlagReplaceTab      [space]
lcc:debugShowGarble     NO
lcc:SQLConnectionWorkstationId      ourWorkstation
lcc:SQLConnectionPacketSize   4096
lcc:SQLConnectionUserId ourUser
lcc:SQLConnectionPassword     ourPassword
lcc:SQLConnectionDataSource   123.123.123.123
lcc:SQLConnectionDataPort     123
lcc:SQLConnectionInitialCatalog     ourDatabase
lcc:SQLQuery       SELECT OWNER,TABLE_NAME,OBJECT_TYPE FROM ourSchema.ourTable WHERE ROWNUM<3 ORDER BY
TABLE_NAME
```

Queries a table with all columns, auto building columns used and column headers

```
lcc:debugMode     NO
```

```
lcc:debugShowLogic      NO
lcc:dataColumnsAuto     YES
lcc:dataColumnsHeadersAuto    YES
lcc:dataColumnsSeparator      [tab]
lcc:dataExportFilename  ctcLinkQuery-PDB130-test-export.txt
lcc:valuesFlagReplaceCRLF     [space]
lcc:valuesFlagReplaceTab      [space]
lcc:debugShowGarble     NO
lcc:SQLConnectionWorkstationId      ourWorkstation
lcc:SQLConnectionPacketSize   4096
lcc:SQLConnectionUserId ourUser
lcc:SQLConnectionPassword     ourPassword
lcc:SQLConnectionDataSource   123.123.123.123
lcc:SQLConnectionDataPort     123
lcc:SQLConnectionInitialCatalog     ourDatabase
lcc:SQLQuery       SELECT OWNER,TABLE_NAME,OBJECT_TYPE FROM ourSchema.ourTable WHERE ROWNUM<3 ORDER BY
TABLE_NAME
```

Queries a table with all columns, auto building columns used and column headers, with SQL commands coming from a file, on an Oracle system

```
lcc:debugMode     NO
lcc:debugShowLogic      NO
lcc:dataColumnsAuto     YES
lcc:dataColumnsHeadersAuto    YES
lcc:dataColumnsSeparator      [tab]
lcc:dataExportFilename  ctcLinkQuery-PDB130-test-export.txt
lcc:valuesFlagReplaceCRLF     [space]
lcc:valuesFlagReplaceTab      [space]
lcc:debugShowGarble     NO
lcc:SQLConnectionWorkstationId      ourWorkstation
lcc:SQLConnectionPacketSize   4096
lcc:SQLConnectionUserId ourUser
lcc:SQLConnectionPassword     ourPassword
lcc:SQLConnectionDataSource   123.123.123.123
lcc:SQLConnectionDataPort     123
lcc:SQLConnectionInitialCatalog     ourDatabase
```

```
lcc:SQLConnectionPlatform     Oracle
lcc:SQLQuery      [lccQueryFromFile:ourSQLQuery-sql.txt]
```

Definitions

Adhoc - on-the-fly value.
XML - eXtensible Markup Language

Modifications

NAME	DATE	MODIFICATION
David Mielcarek	9/11/2009	Created
David Mielcarek	5/11/2011	Updated
David Mielcarek	8/29/2011	Updated
David Mielcarek	2/17/2012	Updated: query Fill-In
David Mielcarek	11/15/2012	Added timeout setting
David Mielcarek	5/15/2013	Added runPostCommand
David Mielcarek	12/10/2014	Modified lcc:exportColumn[...] key notes.
David Mielcarek	9/10/2015	Added lcc:exportColumnCapitalize key.
David Mielcarek	7/19/2016	Added sub-key lccQueryFromFile to key SQLQUERY
David Mielcarek	11/22/2016	Added keys 'SQLConnectionDataPort', 'SQLConnectionPlatform', 'SQLQueryFromFileFromList'
David Mielcarek	4/13/2017	Added Date/Time Flags
David Mielcarek	9/19/2018	Added queryStatementFillIn capability to the dataExportFilename key.
David Mielcarek	20190201	Added lccSQLConnectionCommandTimeout
David Mielcarek	20190506	Added flags to sqlQuery for 'WAYRQQuarters...Year'
David Mielcarek	20191114	Upgraded Oracle, provided 32 and 64 bit versions. Added keys "lcc:valuesFlagReplaceCRLF",

		"lcc:valuesFlagReplaceTab"
David Mielcarek	20191205	Added keys lcc:dataColumnsAuto, lcc:dataColumnsHeadersAuto
David Mielcarek	20200108	Added Tab Delimited Logic Files, see 'lcc:logicPath'. Added Adhoc User Input, see 'lcc:userInput'.
David Mielcarek	20200206	Added Keys 'lcc:recordsFoundHeartbeat', 'lcc:flushMaxRecords'. Upgraded memory management for large downloads.
David Mielcarek	20200219	Added Key 'lcc:exportMaxRecords'
David Mielcarek	20200227	Added support for lcc:queryStatementFillIn within Tab delimited Logic File
David Mielcarek	20200322	Added flag keys for [~WATerms...
David Mielcarek	20200402	Added Key "lcc:dataColumnsQualifyAll"
David Mielcarek	20200420	Added lcc:dataColumnChangeFormat
David Mielcarek	20200518	Added lcc:paramId, lcc:paramValue
David Mielcarek	20200608	Added keys 'lcc:dataColumnsAutoFilterOut', 'lcc:dataColumnsHeadersAutoFilterOut'
David Mielcarek	20200625	Added support for using lcc:SQLQuery and lcc:dataExportFilename multiple times in a single Logic File.
David Mielcarek	20200707	Added Key 'lcc:logicFromFile'
David Mielcarek	20200824	Added Key 'lcc:dataColumnsQualifyAllExcludeEmpty'

End of document