Adabas Data Masking

Data Masking

Version 1.1.1

April 2012

Data Masking

This document applies to Adabas Data Masking Version 1.1.1.

Document ID: DMZ-DOC-111-20120402

Table of Contents

1 Data Masking

Contents		
	Release Notes	Release Notes for Data Masking.
	Getting Started	An introduction of Data Masking.
	What Databases Does Data Masking Support?	Product prerequisites and additional information that you should be aware of before and after installing.
	Installing Data Masking on Windows	Information on how to install Data Masking on Windows platforms.
	Installing Data Masking on Linux/UNIX	Information on how to install Data Masking on Linux/UNIX platforms.
	Obtaining a License Key for Data Masking	Information on how to obtain a license key.
	Using Data Masking	Basic information how to use Data Masking.
	Using GTMapper™	Basic information how to use GTMapper.
	Building Map Files for Standard RDBMs	Contains detailed descriptions of how to build the map files required to use Data Masking with RDBMs.
	Building Map Files for FLAT Files	Contains detailed descriptions of how to build the map files required to use Data Masking with flat files.
	Managing SEED Tables	Contains detailed descriptions of how to manage SEED tables.
	Adding a `WHERE` Clause to a Table to be Masked	Explains how a subset of a table can be masked by adding SQL to the mapping definition.
	Masking Substrings	Explains how to mask substrings in columns containing character datatypes.
	Masking Multi-Record Type Fixed-Width Files	Provides information on how to mask multi-record type files with a fixed width.

Contents		
	Masking Functions and Parameters	Provides detailed information on how to use the masking functions and parameters.
	Managing Primary Keys and Foreign Keys	Explains how to manage primary and foreign keys.
	Managing Large Table Updates or Large Seed Tables	Explains how to manage large table updates and seed tables.
	Appendix 1: Connection Files	Provides information on how to connect to the various supported DB or file systems.

2 Release Notes

These Release Notes summarize the new features and enhancements that are provided with Data Masking Version 1.1.

Availability

Data Masking is available on the following platforms:

- Windows Server 2008 - 64 bit
- Windows 7 Professional Edition - 32 bit
- Windows 7 Professional Edition - 64 bit
- HP-UX 11.i v3 - 64 bit (Itanium)
- AIX 7.1 - 64 bit
- AIX 6.1 - 64 bit
- Solaris 10 - 64 bit (Oracle and FTS)
- SUSE Linux Enterprise Server 11 (z/Linux)
- SUSE Linux Enterprise Server 11 for AMD64 and Intel EM64T (x86-64)
- Red Hat Enterprise Linux 6.0 (z/Linux)
- Red Hat Enterprise Linux 6.0 for AMD64 and Intel EM64T (x86-64)

Prerequisites

The following prerequisites must be met when using Data Masking.

- Data Masking for Adabas uses the Adabas SQL Gateway to access the Adabas database.

3 Getting Started

Software AG understands many testing teams want to avoid installing a heavy infrastructure to mask or obfuscate copies of production data. Some organizations just need a simple solution with a competitive price tag. With this is in mind we have added Simple Data Masking™ to complement our fully functional Fast Data Masking ™ solution. Simple Data Masking™ is a completely separate stand-alone software solution that is low cost, simple to administer, incredibly easy to learn and fast.

The clue is in the name - Simple Data Masking™ is simple-to-use, whilst offering a fast, robust and repeatable method to securing sensitive data. With Simple Data Masking™, you can create and store your masking definitions in spread sheets, and quickly add your own custom masking rules for all database types using just one tool. Within minutes you will be using high-quality, meaningful data that meets full regulatory.

What is Data Masking?

Data masking (also known as de-identification, de-sensitization, obfuscation, scrambling or anonymization) is the process of masking sensitive or personally identifiable information (PII) to ensure that it is secure for use in non-production environments. In recent years, the use of masked data in non-production has become the standard best practice, demanded by several pieces of national and international legislation, including: HIPAA and GLBA (USA), the PCI-DSS and the EU Data Protection Directive.

Failure to comply with these industry regulations has resulted in organizations suffering severe financial penalties, as well damaging their brand reputation and relationships.

What Type of Systems is the Data Masking Solution Suitable for?

Simple Data Masking™ is suitable for organizations who need to obfuscate or mask data for testing, quickly and simply, across multiple small and medium-sized projects.

Enterprise Data Masking™, our enterprise-wide data masking solution, offers a more sophisticated data masking solution for more complicated IT infrastructures. Enterprise Data Masking™, which is powered by our complete test data management suite, Datamaker™, offers a more holistic solution for organizations with large data masking projects, or who wish to undertake intelligent, best practice techniques for masking.

What does Data Masking do?

Simple Data Masking™ provides, in one simple-to-use data masking engine, all of the tools needed to uncover, and then de-identify, scramble and obfuscate your test data.

Current industry regulations and guidelines demand that all data is masked, or deidentified, for use in non-production environments. Non-compliance with standards like HIPAA, the PCI DSS and the EU Data Protection Directive can, and has, resulted in large fines, damage to brand reputation and loss of customer trust, due to data breaches.

4 What Databases Does Data Masking Support?

Data Masking currently supports the following databases:

- Adabas,
- Oracle,
- DB2 (UDB and z/OS),
- DB2 400 for iSeries,
- Microsoft SQL Server,
- Teradata,
- MySQL,
- Sybase,
- PostgreSQL,
- SQLAnywhere (v10.00.0.1.x onwards),
- Informix,
- Intersystems Cache,
- VSAM/ISAM or any Shadow Direct enabled data source and
- flat files.

How is the Software Installed and what are the System Requirements?

Simple Data Masking™ requires a Java Runtime Environment (JRE) and can be run on Windows, Linux, UNIX, z/OS platforms. Should there be a problem with Java, the Simple Data Masking™ contains a \jre folder in which you will find a suitable version of JRE.

All masking rules are stored in spreadsheets. Standard seed tables are shipped with and can be easily added and amended to the product.

How Does Data Masking Work?

A series of intuitive and straightforward rules are defined to mask and anonymize the data. The collection of rules includes:

- Using seed tables
- Multi-table column values
- Hashing
- Replacements

- Custom Functions
- Translations
- Offset dates
- Substitution
- Credit Cards
- Random ranges
- Random Text
- Numeric variances

5 Installing Data Masking on Windows

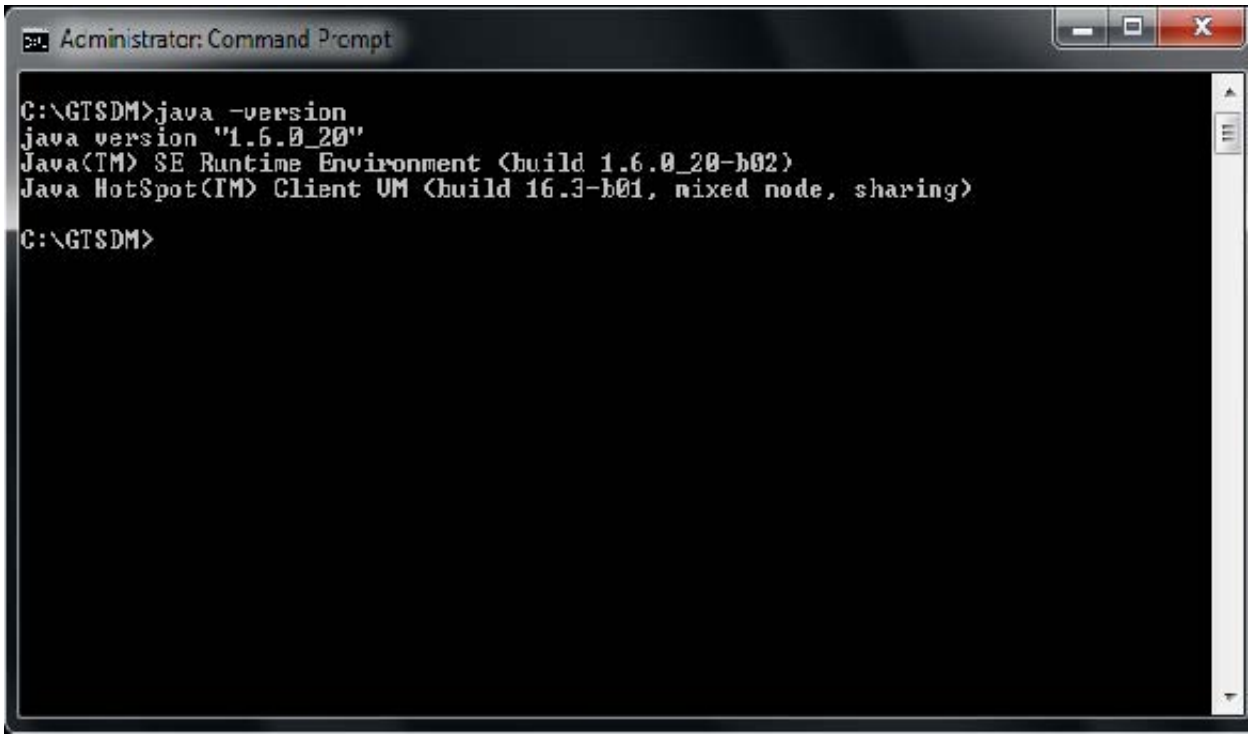
Run the installer setup_gtsdm.exe on the PC you wish to run Data Masking on.

Make sure you have connectivity from the PC to any databases you will be masking. The installer will create a folder called: \GTSDM.

In this folder there will be a subfolder called \GTSDM\SEEDTABLES.

The SEEDTABLES folder contains seed data files. Make sure you have a Java Runtime 1.6 or higher installed.

To check the version of Java you have, open a command prompt and issue: `java -version`.



6 Installing Data Masking on Linux/UNIX

Make sure you have a Java Runtime 1.6 or higher installed.

To check the version, of Java you have, open a command prompt and issue: `java -version`.

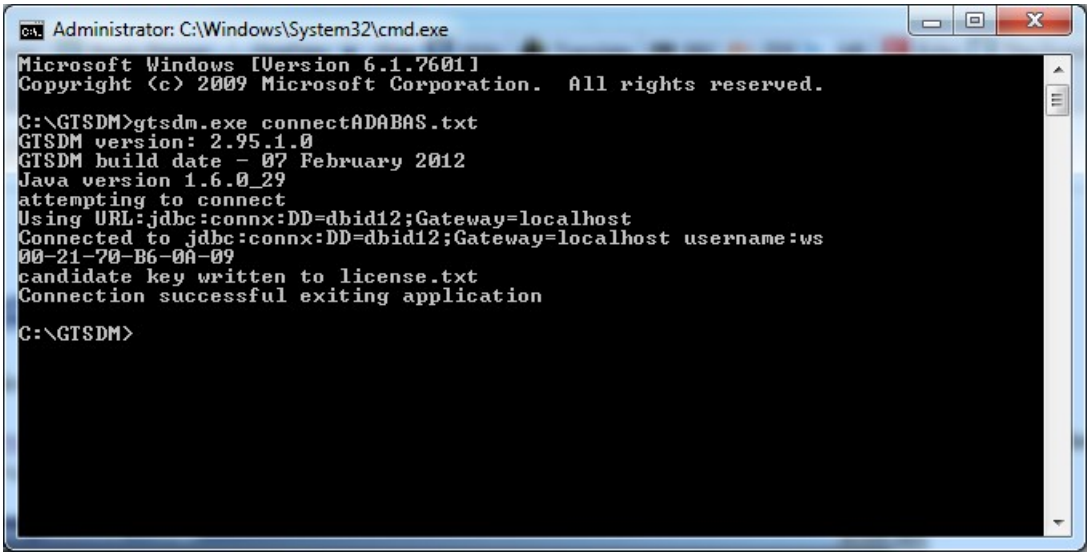
Extract the archived file in a chosen directory (example: /home/GTSDM).

Ensure you have the rights and privileges to access the files and folders in this directory.

7 Obtaining a License Key for Data Masking

In order to obtain a license key for Data Masking, you will need to follow these steps:

1. Generate a candidate key by running the file GTSDM.exe `<connection profile>`.txt, as shown below:



   You will then need to send the candidate key to Software AG to acquire a license key.

2. Once you have submitted a candidate key, you will receive a .txt file containing your Data Masking product license key. To activate the license, add the following parameter to your `connection profile.txt`:

   `License=<license key>` as shown below.

Note: The product kit as shipped by Software AG has a license which is valid for a maximum of 4 weeks and has a maximum usage of 5000 rows per table included.

8 Using Simple Data Masking™

Linux

Simple Data Masking is a single executable .jar file that needs to be run from the terminal. The GTSDM executable .jar requires two files to run: the first file contains the connection parameters used to connect to the database to be masked; the second contains a list of the columns to be masked and the type of masking to be performed.

E.g. `java -jar gtsdm.jar connectoracle.txt maskoracle.csv [options.txt]`

When the product is installed a sample of connection profile files with a suffix of .txt are included, for example:

- ConnectOracle.txt
- ConnectVSAM.txt

To connect to your database copy one of these text files and rename it to a meaningful name, for example: ConnectMyOracleHuw.txt

The second file contains the obfuscation mappings. A few samples mappings are included, for example:

- MapOracle.csv
- MapVSAM.csv

You can copy one of these files and start editing using CSV Editor tools or a text editor.

Windows

Simple Data Masking is a single executable that needs to be run from a command prompt. The GTSDM executable requires two files to run: the first file contains the connection parameters used to connect to the database you wish to mask; the second contains a list of the columns to be masked and the type of masking to be performed.

E.g. `gtsdm.exe connectoracle.txt maskoracle.csv [options.txt]`

When the product is installed, a sample of connection profile files, with a suffix of .txt are included, for example:

- ConnectOracle.txt
- ConnectVSAM.txt

To connect to your database, copy one of these text files and rename it to a meaningful name, for example: ConnectMyOracleHuw.txt

The second file contains the obfuscation mappings. A few samples mappings are included, for example:

- MapOracle.csv
- MapVSAM.csv

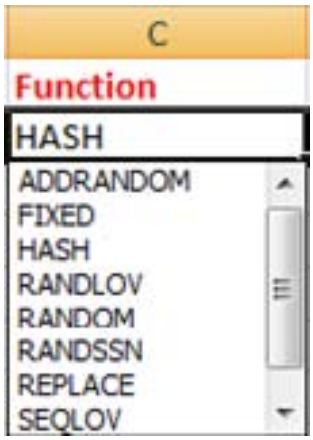
You can copy one of these files and start editing using Microsoft Excel or a text editor.

If you would prefer to use a spread sheet to edit CSV files, an .xls file called `BuildMap.xls` is provided which can be used to create csv map files for you.

To use this, open `BuildMap.xls` in Excel:

Once you have made edits to the workbook you can save the file in a CSV format.

Note: There is a quick drop-down of available masking functions, which can be used to quickly apply randomisations to specific columns. This can be accessed by right-clicking on the appropriate cell and selecting **Pick from a Drop-Down List**:



When you have finished your definitions you can use **Save As** and choose CSV as the format.

You will be prompted as follows:

Click **OK**.

Click **Yes**.

When you exit you will receive the message:

Click **No** as you have already saved it.

9 Using GTMapper™

The GTMapper™ allows you to create and execute your masking definitions in one clean, easy-to-use program.

The GTMapper.exe can be found in your c:\gtsdm folder. It can also be found as a shortcut in your Start Menu as a subfolder of Grid-Tools.

Note: The Build Version and Date are displayed in the top left corner for reference. This information can also be found in the GTSDM.exe and GTSDM.jar files in GTSDM folder.

Once opened, GTMapper™ will present you with the following window. This is the window in which you are able to create and execute all your masking definitions. Along the top of the window you will notice three tabs; **Connection Parameters**, **Scramble Functions** and **Scramble Options**.

Working with the Connection Parameters Tab

The first window you see will be the **Connection Parameters** tab. The left-hand panel offers a list of available connections. The connection details are shown to the right.

Note: For a connection file to be visible in the GTMapper™ Connection Parameters tab, it must be saved as a `connect<filename>.txt` file, i.e. `connectORACLE.txt`

iSeries users can click on the default connection file `connectdb2400.txt` and change the following parameters:

1. Username – iSeries username
2. Password – iSeries password
3. Database = Default Schema = iSeries Library or SQL Schema that contains the tables to be masked

Once your connection is set, click the Connect button in the bottom right-hand corner of the window to establish your connection to the database selected.

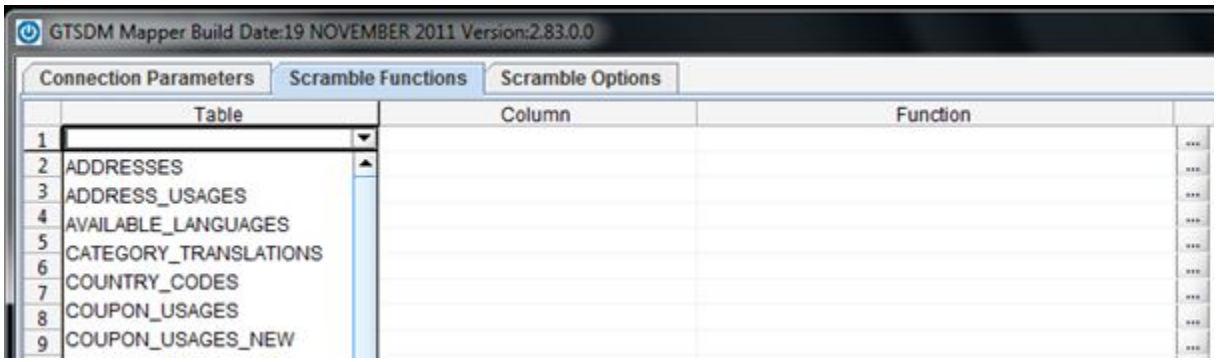
Working with the Scramble Functions Tab

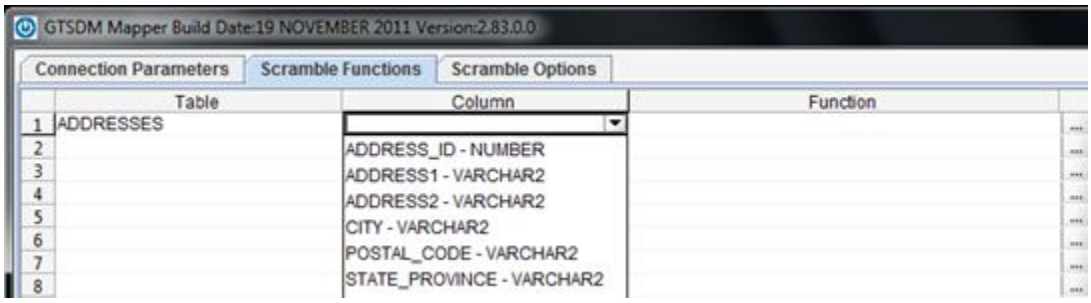
In **Scramble Functions**, you can create a new map, or edit an existing one.

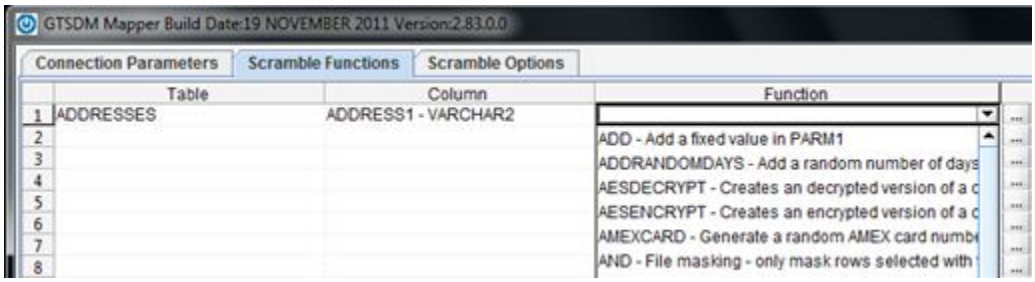
In order to open an existing CSV file, select **Open Mapping** in the bottom right-hand corner of the window and select the previously created file from the GTSDM folder.

The **Scramble Functions** tab gives users the ability to see and select; any table in the default schema, any column in the selected table and any applicable masking functions for the selected column.

To select from any of the possible Table, Column or Function options, left click on the desired cell and choose from the drop-down menu.







When saving your masking definition, GTMapper™ also gives users the option to Save & Run directly from this screen, using the **Save & Run** button in the right-hand corner.

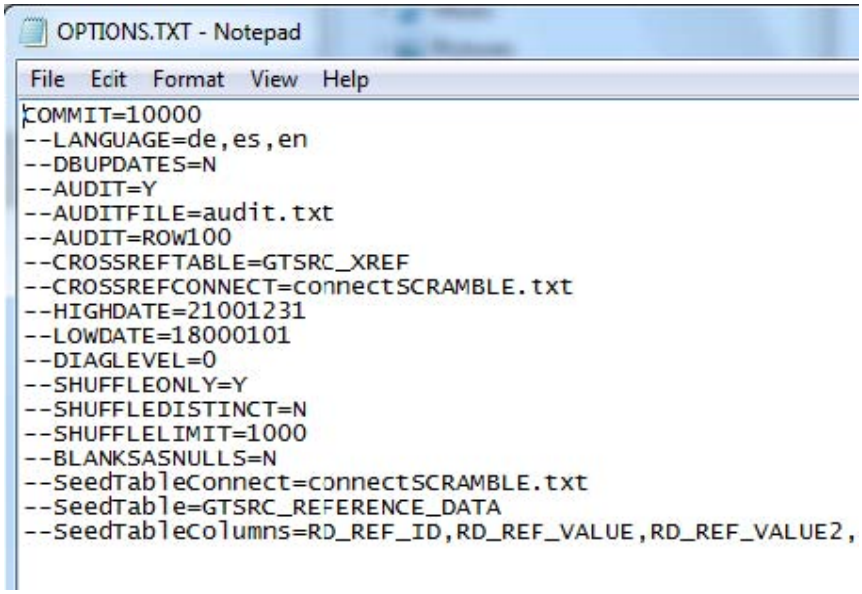
Save the file as a CSV. The details of the mask are shown in the screen shown below.

Note: db2400 and db2 for iSeries users are discouraged from this as it may cause the transfer or large amounts of data from iSeries to their Windows machine. These users should consider using the **Save & Run** button when they have small amounts of data.

Working with the Scramble Options Tab

The **Scramble Options** tab allows you to set some very important options for the masking process. These include additional parameters to control the run, for example, you may wish to adjust the commit frequency.

The options available are explained below:



Audit

The Audit file contains the table name, the column name(s) that constitute a unique row for the table, the values for these column(s), and the old and new values for columns that are being masked. The Audit options are as follows:

COMMIT=*nnnn*	Commit after *nnnn* rows for each table to be masked.
AUDIT=ALL	All rows will be audited.
AUDIT=ROW*nnn*	Where *nnn* is equal to the number of rows that will be audited, for example ROW*1000* will audit the first 1000 rows.
AUDIT=SAMPLE*nnn*	Where every *nnn* rows will be displayed, for example SAMPLE*100* will audit every 100th row.
AUDITFILE=	The filename. The format will be comma separated.
AUDITONLYCOLUMNS=	Allows you to provide a comma separated list of column names to be audited, for example: EMPLOYEE.FIRST_NAME, EMPLOYEE_ADDRESS.CITY
AUDITVALUES=	N = do not show old and new values in AUDIT file. The default setting is to show both.

Cross-Reference

CROSSREFTABLE=	gtsrc_xref - the name of the table to read and write cross reference data to.
CROSSREFCONNECT=	This refers to a file containing the connection information of the schema that contains the cross reference table.
CASEINSENSITIVEXREF=	Make comparisons case insensitive (For Cross-Reference)
TRIMMEDXREF=	Trim values before comparing (For Cross-Reference)

	A	B	C	D	E	F
1	TABLE	UNIQUE COLUMNS	UNIQUE COLUMN VALUES	MASK COLUMN	OLDVALUE	NEWVALUE
2	hotels2	id	1415	address	Aigburth hall road	Bateman Street
3	hotels2	id	3679	address	Fairfield street	Redhouse lane
4	hotels2	id	3697	address	Rupert hill	Caryl street
5	hotels2	id	3698	address	Steble street	Chatham street
6	hotels2	id	3699	address	Hawke street	Grove Street
7	hotels2	id	3700	address	Roscoe street	Cemetery Road
8	hotels2	id	3868	address	Market street	Muirhead avenue
9	hotels2	id	4014	address	Oldham street	Beechwood road
10	hotels2	id	4077	address	Sweeting street	Taggart avenue
11	hotels2	id	4412	address	Waterhouse street	North sudley road

Each database type will have a starter connection .txt file. Copy the appropriate file for your RDBMS to your own file name and begin editing this file. Sample files for different RDBMS are shown in the **Building Maps Files for Standard RDBMS** section.

Date

CDATE=	Override todays date for the purposes of date calculation functions, for example, DOB (format: YYYYMMDD).
HIGHDATE=	Override the highest date that offset date functions will process, for example, dates later than 22000101 will be ignored.
LOWDATE=	Override the lowest date that offset date functions will process, for example, dates earlier than 18000101 will be ignored.

Languages

GTSDM currently supports 3 languages for messaging – English, German and Spanish.

When GTSDM starts, it checks the current default locale - if the language is supported (for example, one of the three above), it will process messages in that language. If the language is not supported, it will default to English, unless set in the Options file.

You can override the locale language by altering the language option as follows:

LANGUAGE=	en (English), de (German) or es (Spanish).

Mapper

XPATH ELEMENT	The location of the XML data you wish to be masked.

Parallelism

PARALLEL=4	Enables users to separate a *maximum* of 4 masking threads.

SDM will assign a separate thread for each table in a CSV if there is more than one. However, more typically, a CSV will have just one table but be split using WHERE clauses.

For example, a CSV using the `WHERE` clauses below would have 4 spilts:

- WHERE, CUSTID<100 ....
- WHERE, CUSTID BETWEEN 100 AND 200 ....

- WHERE, CUSTID BETWEEN 200 AND 300 ....
- WHERE, CUSTID>100

Seed Tables

SEEDTABLECONNECT=	connectscramble.txt - the name of the connection file to get seed data from.
SEEDTABLE=	gtsrc_reference_data - the name of the table to get the seed data from.
SEEDTABLECOLUMNS=	Comma separated list of the columns in SEEDTABLE.

Shuffle

SHUFFLEDISTINCT Y/N	Y selects distinct values for the shuffle creates. N is default and selects all values.
SHUFFLELIMIT *n*	Only select *n* values for the shuffle.
SHUFFLEONLY Y/N	Y does not update the database, instead it just produces the shuffle files or database shuffle values.

Other

BADDATESTRING=	For DOB/DOD on dates stored in character fields. Specify the date to replace the unparseable data as YYYY/MM/DD.	
BLANKSASNULLS=Y	Sets all blank values as Nulls. **Note:** Nulls are ignored when RANDLOV, SEQLOV or SHUFFLE with joined columns - to maintain consistency between columns.	
CASEINSENSITIVESEED=	Y - Makes the search on rd_ref_value column, using RANDLOV1 function case insensitive **Note:** This option is specific to the RANDLOV1 function.	
DBUPDATES=	N	Run in simulation mode.
	S	Creates SQL file `<table name>`_UPDATES.sql **Note:** DBUPDATES=S option is only available for non-db2 databases with unique or primary key columns.

	P	■ Pre-Step: SQL file that is executed prior to masking that typically drops triggers and FK constraints ■ Post-Step: SQL file that is executed after masking that typically re-creates triggers and FK constraints
	V	Do not perform database update, just validate CSV map.
DB2BATCHUPDATE=	N (Default). If Y, use fast batched updates rather standard 'update where current of' cursor method (DB2 only).	
DIAGLEVEL=	0,1 or 2. Debug info will be output according to value.	
EMPTYASNULL=Y	Treats all blanks or spaces as null values **Note:** Nulls are ignored when RANDLOV, SEQLOV or SHUFFLE with joined columns to maintain consistency between columns.	
LOADALLSEEDDATA=	N (default) - Loads all seed data into Java memory for the RANDLOV1 function, irrespective of the distribution of rd_ ref_values in the table to be masked.	
ORDERBY=	Y (Default) - Decides whether or not selected data will be ordered by PK columns. You may wish to turn off for VMS. **Note:** If you select ORDERBY=N, the audit file may not show the masked records in an ascending order of PKs. This does not allow the restartability of SDM after a failure.	
PROCESSCOUNT=	Limits the number of rows processed in a run, allowing you to check the mapping without doing a full masking run.	
RELAXNONINDEXVALID=	XREF on non varchar columns of no PK/UK tables.	
WHEREASSUBSET	Y (default) - use as WHERE subset on flat files. N - use WHERE as criteria for masking and output all records.	
USERFAASINDEX=	Use RFA in WHERE clause of the update SQL (VMS DB ONLY).	
USERRNASINDEX=	For non-indexed tables, use RRN function to get row identifier - then combine with DB2BATCHUPDATE to use fast method (DB2400 ONLY).	

10 Building Map Files for Standard RDBMs

The map files contain a list of tables, columns and the masking rules you wish to apply.

	A	B	C	D	E	F	G	H	I	J	K
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls	Dateform	Cross Reference	Notes
2	ADDRESS	POSTAL_CODE	RANDLOV	ukpostcode.1.txt							
3	CREDIT_CARDS	CARD_NUMBER	GENCARD	T							
4	ADDRESS	CITY	RANDLOV	ukpostcode.2.txt							
5	ADDRESS	STATE_PROVINCE	RANDLOV	ukpostcode.3.txt							
6	PEOPLE	LAST_NAME	RANDLOV	lastname.txt							

Simply, you build up a list of columns and assign a masking function to each column.

Note: Please make sure that the first row of the CSV file is kept in the same format with the same column headings. Some functions may require additional parameters which you add in on the same row as the column in which you wish to perform the mask.

The **KeepNulls** entry specifies whether or not to retain any null values in the database, or to replace them with masked values. The default is to retain null values. If you wish to replace all values in the column, you need to enter N in this column.

Note: For certain functions that add random values to existing values, any null values are always retained.

The **Dateformat** column is used to specify the format for any dates that are held in the database in character fields.

The **Preformat** column is used to specify the format of the data prior to it being sent for masking to a substring.

Note: The **Preformat** column currently only supports the USZIP and USZIP+4 functions. It is particularly when masking multiple columns using seed data (RANDLOV1).

Note: Every valid-looking USZIP code should get converted to a USZIP5 format:

- Case 1 : - Zip code is 3 digit [append 2 zeros on left] - 983 > USZIP5 > 00983
- Case 2 : - Zip code is 4 digit [append 1 zero on left] - 4789 > USZIP5 > 04789
- Case 3 : - Zip code is 5 digit [leave as-is] - 12345 > USZIP5 > 12345
- Case 4 : - Zip code is 7 digit [append 2 zeros on left and remove 4 digits from right] 9831234 > 00983
- Case 5 : - Zip code is 8 digit [append 1 zero on left and remove 4 digits from right] 47891234 > USZIP5 > 04789 [remove 4 digits from right]
- Case 6 : - Zip code is 9 digit - 123459999 > USZIP5 > 12345
- Case 7 : - DEFAULT [Leave as-is] - 1 > 1 or NULL > NULL

The **Cross Reference** function allows you to consistently change a value so that it will be changed to the same value across databases and connections. The cross reference function will maintain a table of old and new values. Before masking a value, Simple Data Masking™ will check to see if the value has been masked before. If so, it will be replaced with the prior mask. If not, Simple Data Masking™ will build a new mask and also populate the cross reference table.

For example, if you have an Ingres table called CUSTOMER with a column CITY and 'New York' is changed to 'Chicago', the table gtsrc_xref will now contain the values:

US CITY New York Chicago

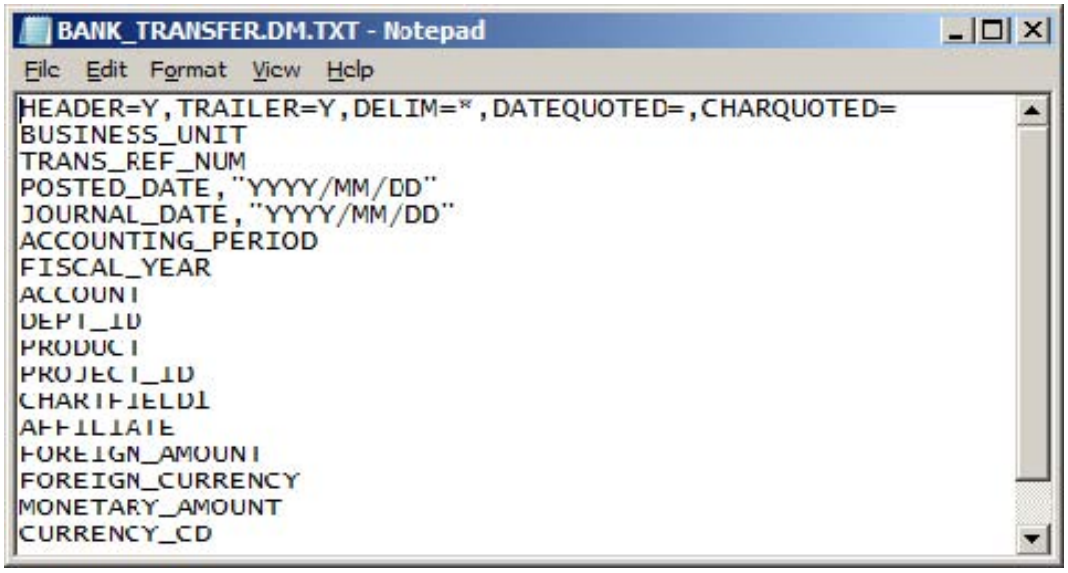
If you now have an Oracle table called PERSON with a column called ADDRESS3 and it has the same cross reference identifier and a value of 'New York', it will be changed to 'Chicago'.

Note: You should not use the Shuffle function in maps with Cross-References.

See the *Options File* section above to see how to set up the cross reference functionality.

11 Building Map Files for FLAT Files

In addition to the normal CSV file used to define the mapping, Simple Data Masking™ allows you to create an additional file containing the layout of the file to be masked. The normal suffix of this file is DM.TXT.



The file's first row gives general details about the file. The options are:

HEADER=Y/N	The file has a header.
TRAILER=Y/N	The file has a trailer.
DELIM=specific character	For comma use <COMMA>, for tab use <TAB>
DATEQUOTED=Y/N	Any dates will be double quoted.
CHARQUOTED=Y/N	Any character columns will be character quoted.

See the *Datamaker™ User Guide* for additional details on this file layout descriptor.

The additional lines contain the name of each of the columns. The name must match with the COLUMN name in the map.csv (the mapping file). If the column is a date field then include the date format in double quotes, see the example above.

Table	Column	Function	Parm1	Parm2
BANK_TRANSFER	TRANS_REF_NUM	FIXED	Masked	
BANK_TRANSFER	ACCOUNT	INTRANGE	1000000	9000000
BANK_TRANSFER	FOREIGN_AMOUNT	INTRANGE	100	1000

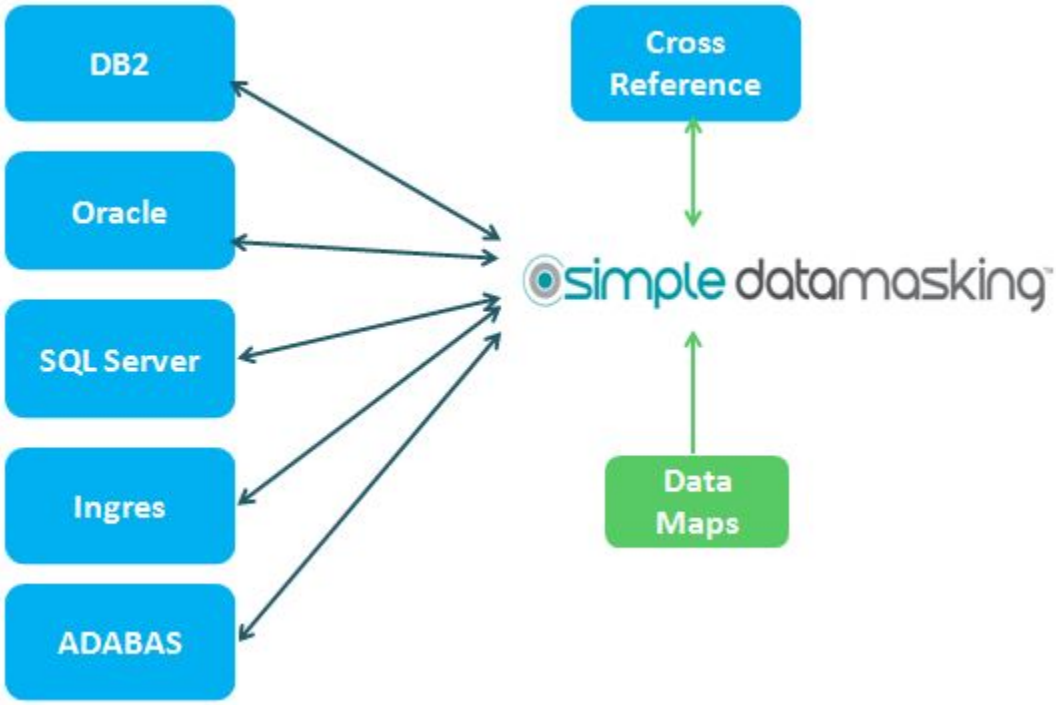
You can then assign the normal masking functions in the same way as for the RDBMs maps. The functions WHERE and SQLFUNCTION will not apply for flat file masking, however cross reference will.

Cross Reference Table

To use the cross reference functionality you will need to create a table in one of your connections that has the following structure:

```
CREATE TABLE gtsrc_xref (rx_ref_id varchar2 (254) NOT NULL,rx_new_value varchar2 ↩
(254));
ALTER TABLE gtsrc_xref ADD CONSTRAINT gtsrc_xref_pk PRIMARY KEY ↩
(rx_ref_id,rx_old_value);
```

If you clear this table down then any existing cross reference mappings will be re-built the next time you run the mask.

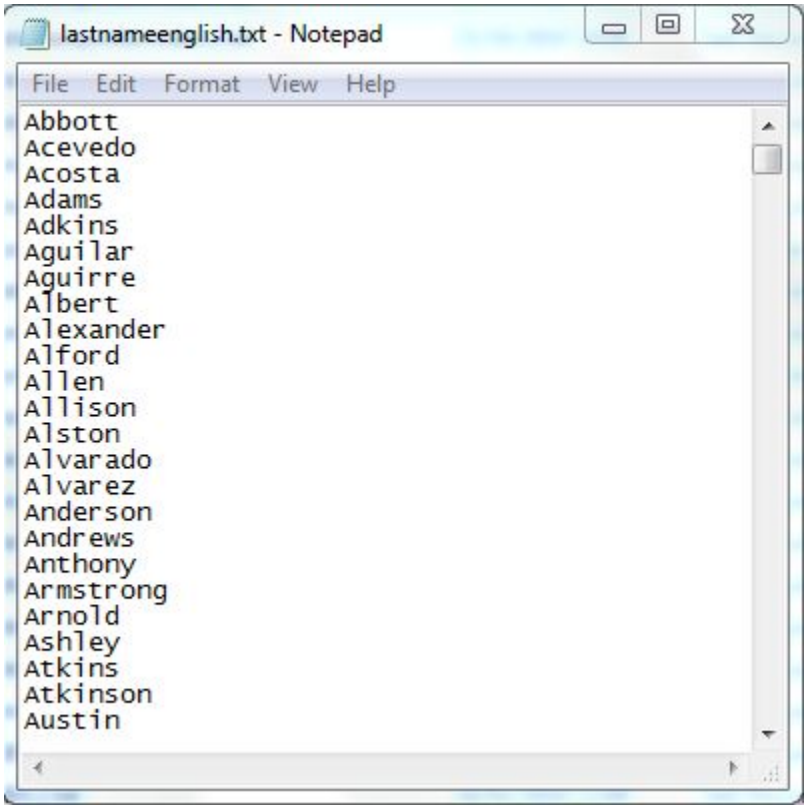


12 Managing SEED Tables

The SEEDTABLES folder contains seed data in editable text files. Each of these files contains a list of values which will be used for masking the columns, when referenced.

australianpostalcodes.1.txt	15/10/2010 10:04	Text Document
australianpostalcodes.2.txt	15/10/2010 10:04	Text Document
bankaccountno.txt	02/06/2010 12:25	Text Document
banknames.txt	19/05/2010 14:43	Text Document
companies.txt	04/05/2010 17:18	Text Document
computergames.txt	16/02/2010 10:01	Text Document
country.txt	16/02/2010 10:05	Text Document
creditcard.txt	16/02/2010 10:06	Text Document
currencycode.txt	16/02/2010 13:53	Text Document
dayofweek.txt	16/02/2010 10:06	Text Document
emailaddresses.txt	19/05/2010 15:03	Text Document
energycompanies.txt	05/05/2010 11:17	Text Document
femalenames.txt	16/02/2010 13:55	Text Document
firstnamefemaleamerican.txt	23/03/2010 10:22	Text Document
firstnamemaleamerican.txt	23/03/2010 10:32	Text Document
firstnamemaleamerican1.txt	16/03/2010 16:31	Text Document
firstnames.txt	04/05/2010 14:32	Text Document
germanpostalcodes.txt	16/02/2010 13:57	Text Document
icd10.1.txt	16/02/2010 13:47	Text Document
icd10.2.txt	16/02/2010 13:49	Text Document
indiancities.txt	16/02/2010 13:49	Text Document
lastnameenglish.txt	16/02/2010 13:51	Text Document
lastnameindian.txt	16/02/2010 13:58	Text Document

Contents of the sample seed file `lastnameenglish.txt` are as shown below.



To use this seed file to mask the column(s) in your database, use the seedfile name with an appropriate masking function (RANDLOV or SEQLOV). For example, to mask the column LAST_NAME of the table PERSON, the CSV file entry might be:

A	B	C	D	E	F	G	H
Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
PERSON	LAST_NAME	RANDLOV	lastnameenglish.txt				N

Using Seed Data Across Multiple Columns in a Table

Note: RANDLOV or SEQLOV should be used as the masking functions for Seed Data.

When using seed data across multiple columns, users MUST use the options file to specify the seed table connect, table and columns they wish to use.

In this case, you should reference and / or create files in the format `name.1.txt`, `name.2.txt` etc.

For example, table hotels2 has 3 address columns: street name, area and postcode. These can be consistently masked by referencing the parameters `ukpostcode.1.txt`, `ukpostcode.2.txt` and `ukpostcode.3.txt`.

Note: Each seed table must have the same number of values in order to perform mask.

Table	Column	Function	Parm1	Parm2
customer	cust_cc	GENCARD		
customer	cust_shor	RANDLOV	femalenames.txt	
hotels2	address1	RANDLOV	UKPOSTCODE.1.TXT	
hotels2	address2	RANDLOV	UKPOSTCODE.2.TXT	
hotels2	postcode	RANDLOV	UKPOSTCODE.3.TXT	

In other words the corresponding random value is used from the 3 files for each row to be masked in the table.

13 Adding a WHERE Clause to a Table to be Masked

A subset of a table can be masked by adding SQL to the mapping definition.

In the following example, table ADDRESS will only have those rows where `ADDRESS_2 = 'Addyston'` masked.

To do this, leave the column field blank and enter `WHERE` as the function, Parm1 will be the SQL to use on the table. The other row or rows for the table will specify the specific masking to apply to the table.

A	B	C	D	E	F	G	H
Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
ADDRESS	ADDRESS_2	RANDLOV	ukpostcode.1.txt				N
ADDRESS		WHERE	ADDRESS_2='ADDYSTON'				N

14 Masking Substrings

It is also possible to mask substrings in columns containing character datatypes. In order to do this, enter values for BOTH the "Substr start" and "Substr length" columns in the mapping csv.

Note: The values must both be integers > 0 (positioning starts at the value 1, not at 0).

Masking function should not be `SHUFFLE`, `WHERE`, `AND`, `OR`, `DELETE` or `TRUNCATE`.

Masking Substrings That Differ in Length from Value

- If the substring specified does not exist in the column to be masked (the column value is too short), the column will be padded with blanks, then masked.

  Example: If varchar column contains "abc", and your substring starts from position 5 for 2 characters using FIXED (1"2"), then the masked value will be "abc12".

  Note: If the column is null and keepnulls=y applies, then the column will not be masked.

- If the masking value is bigger than the substring specified then it will be truncated.

  Example: If a column contains "abcdefghi" and you attempt to mask from position 2 for 2 characters using ("FIXED 123"), then the masked value will be "a12defghi"

- If the masking value is smaller than the substring specified, then it will be padded with blanks.

  Example: If a column contains "abcdefghi" and you attempt to mask from position 2 for 4 characters using FIXED ("1"), then the masked value will be "a1 ghi".

- If you specify a cross-reference, then the old and new values used to read and update the xref table will be the full column contents (not just the substring).

  Example: To mask characters 13 to 26 of column RAWDATA in table ORIGINAL with "7777777777777", the mapping csv row would be:

  ```
  originalpos,rawdata,FILL,7,,,,,,,,,,13,13
  ```

  or

  ```
  originalpos,rawdata,FIXED,7777777777777,,,,,,,,,,13,13,
  ```

  In the audit, the function name will be suffixed with the start:length parm, "FIXED(13:13)".

I	J	K	L	M	N	O
DateFormat	Cross Reference	Override SQL	Unique Columns	Xpath Element	Substr Start	Substr Length

Applying Multiple Functions to the Same Column

When applying multiple functions to the same column, the functions should be listed in consecutive rows in the mapping csv, as shown below:

A	B	C
Table	Column	Function
PAYMENT_OPTI	ACCOUNT_NUM	SEQNUMBER
PAYMENT_OPTI	ACCOUNT_NUM	FIXED

There is no limit to the number of masking functions performed on each column, however, KeepNulls should be set the same for all of these functions.

Notes:

1. All functions MUST HAVE a "substr start" and "substr length" set. Substr specifications for different functions can refer to the same character positions.
2. If using cross-references, they can only be set for the last function used on each column.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls	DateFor	Cross R	Overric	Unique	Xpath E	Substr start	Substr len
PAYMENT_OPTI	ACCOUNT_NUMBER	SEQNUMBER	10000000001				N						7	13
PAYMENT_OPTI	ACCOUNT_NUMBER	FIXED	0				N						7	1

In the example above, the mapping csv shows that a mask of column ACCOUNT_NUMBER in table PAYMENT_OPTIONS with a zero padded sequence in positions 7 to 16, expressed as:

```
caaccounts,numberx,SEQNUMBER,1000000001,,,,,,,,,,7,10,
caaccounts,numberx,FIXED,0,,,,,,,,,,7,1,
```

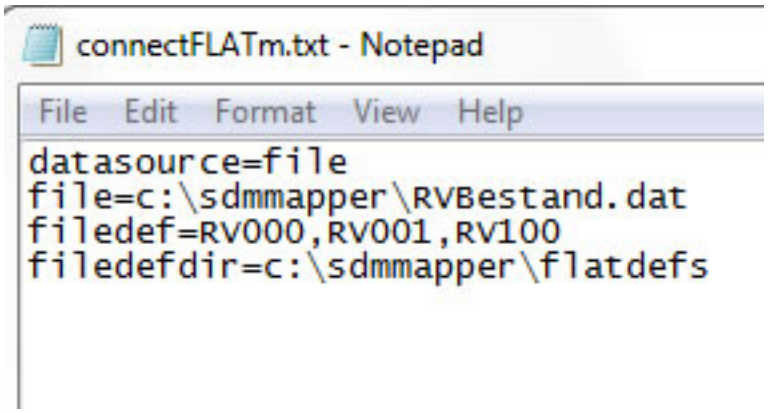
Note: SEQNUMBER returns a left-aligned non-padded number, so the sequence is started at 1000000001 and the leading digit is then set to 0.

In the audit report, the function names will be suffixed with start:length parameters and concatenated together, for example, "SEQNUMBER(7:10) FIXED(7:1)"

Note: If cross-referencing is specified, the value used to lookup and update the cross reference table is the whole column value, not a substring of the value.

15 Masking Multi-Record Type Fixed Width Files

When masking multi-record type fixed width files, the path to the definition files is created by prefixing each `filedef` name with the `filedefdir` and suffixing it with dm.txt. For example (using the connection file below): c:\sdmmapper\flatdefs\RV000.dm.txt



The first `filedef` in the list can specify options which apply to the whole file - HEADER, TRAILER, DATEQUOTED, CHARQUOTED, LINETERM or NULLIND.

Note: None of the options above can be specified in other filedefs. For multi-format masking, all record definitions must have DELIM=FIXED specified.

A record type is identified by the `RECTYPE` parameter on the first line, for example, `RECTYPE=01_SAZART_X3:"000"`. This parameter names a field within the record and gives a literal value which must be matched for that record type to apply. Field definitions consist of the field name, length and, optionally, a field datatype (Date or Numeric, not case sensitive) and mask value (a mask is assumed to be a date unless show to be something else). The field definition must follow the format below:

```
04_ERSTELLDATUM_D10, 10, Date:"DD.MM.YYYY"
```

Where `04_ERSTELLDATUM_D10` is the field name, which is 10 bytes long and contains a date in the format `"DD.MM.YYYY"`

Note: A datatype must be followed by a colon and mask value in double quotations.

16 Masking Functions and Parameters

ADD

Add a fixed value in Parm1. The `ADD` function will also add the fixed value in Parm1 to dates in a character field.

Required parameters:	Parm1.
Applies to database types:	Numeric.
Example:	ship_to_address_ID in table orders will have a value of 5 added to existing.

A	B	C	D	E	F	G	
Table	Column	Function	Parm1	Parm2	Parm3	Parm4	Ke
ORDERS	SHIP_TO_ADDRESS_ID	ADD	5				N
							N

ADDPERCENT

Adds a fixed percentage value in Parm1 to the original value.

Required parameters:	Parm1.
Applies to database types:	Numeric.
Example:	customer_ID in table orders will have a value of 10% added to existing value.

A	B	C	D	E	F	G	H
Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
ORDERS	CUSTOMER_ID	ADDPERCENT	10				N

ADDRANDOM

Adds a random value between Parm1 and Parm2 to the existing value.

Required parameters:	Parm1 and Parm2.
Applies to database types:	Numeric.
Example:	unit_price in table order_items will have a value between -4 and 4 added to the existing value.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	TRACK MONTHLY	NUM_CARDS	ADDRANDOM	4	-4			N
3								

ADDRANDOMDAYS

Adds a random number of days between Parm1 and Parm2 to the existing value.

Required parameters:	Parm1 and Parm2.
Applies to database types:	Date, Datetime.
Example:	The test_date column in table credit_cards will have between 10 and 100 days taken off the existing value for that column.

	A	B	C	D	E	F	G	H	I	J
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls	Dateform	Notes
2	credit_cards	test_date	ADDRANDOMDAYS	-100	-10					
3										
4										
5										
6										

ADDRANDOMHOURS

Adds a random number of hours between Parm1 and Parm2 to the existing value.

Required parameters:	Parm1 and Parm2.
Applies to database types:	Date, Datetime.
Example:	creation_date will have between 6 and 9 hours added to existing value.

	A	B	C	D	E
1	Table	Column	Function	Parm1	Parm2
2	SHIPPING_OPTIONS_BASE	CREATION_DATE	ADDRANDOMHOURS	6	9
3					

ADDRANDOMMINUTES

Adds a random number of minutes between Parm1 and Parm2 to the existing value.

Required parameters:	Parm1 and Parm2.
Applies to database types:	Date, Datetime.
Example:	creation_date will have between 30 and 45 minutes added to existing value.

	A	B	C	D	E
1	Table	Column	Function	Parm1	Parm2
2	SHIPPING_OPTIONS_BASE	CREATION_DATE	ADDRANDOMMINUTES	30	45
3					

ADDRANDOMSECONDS

Adds a random number of seconds between Parm1 and Parm2 to the existing value.

Required parameters:	Parm1 and Parm2.
Applies to database types:	Date, Datetime.
Example:	creation_date will have between 4 and 13 seconds taken off the existing value.

	A	B	C	D	E
1	Table	Column	Function	Parm1	Parm2
2	SHIPPING_OPTIONS_BASE	CREATION_DATE	ADDRANDOMSECONDS	4	13
3					

ADDRANDOMYEARS

Adds a random number of years between Parm1 and Parm2 to the existing value.

Required parameters:	Parm1 and Parm2.
Applies to database types:	Date, Datetime.
Example:	CREATION_DATE in table OPTIONS_BASE will have between 6 and 9 years added to the existing value.

	A	B	C	D	E
1	Table	Column	Function	Parm1	Parm2
2	OPTIONS_BASE	CREATION_DATE	ADDRANDOMYEARS	6	9
3					

AESDECRYPT

Creates a decrypted version of a column based on the key used in Parm1. The column must have been previously encrypted using the **AESENCRYPT** function with same Parm1 value.

Required parameters:	Parm1, which is the encryption key. **Note:** The encryption key can not be more than 16 characters in length.
Applies to database types:	Character.
Example:	The value `17 Windmill St` is decrypted to `4e 23 11 6a 61 30 27 0d 10 f1 08 56 fc dc 13 dc`.

AESENCRYPT

Creates an encrypted version of a column using an AES (Advanced Encryption Standard) algorithm and the key in Parm1.

Note: The output value will be longer than the original.

Required parameters:	Parm1, which is the encryption key. **Note:** The encryption key can not be more than 16 characters in length.
Applies to database types:	Character.
Example:	The value `17 Windmill St` is encrypted to `33 6f 53 90 f9 7b c3 aa 09 e9 11 aa 2b d9 ba bd`.

AND

The `AND` function, in conjunction with `WHERE`, allows you to restrict your obfuscation to only certain rows. For example, you can mask male customers differently based on the `MALE_NAMES AND PERSON_TYPE_CODE` columns. The `ADD` function must be used on a separate row to the WHERE function.

Note: The `ADD` function is for masking flat files only as there is no SQL file.

Required parameters:	Parm1. This is an SQL `WHERE` clause.
Example:	

	A	B	C	D	E
1	Table	Column	Function	Parm1	Parm2
2	PERSONS		WHERE	GENDER LIKE 'M'	
3	PERSONS		AND	PERSON_TYPE_CODE LIKE 'CUST'	
4	PERSONS	FIRST_NAME	RANDLOV	malenames.txt	
5					

CHARHASH

Converts a character hashed value from the input (PARM1 must be set to the method MD2, MD5, SHA-1, SHA-256, SHA-384, SHA-512).

Required parameters:	Parm1.
Applies to database types:	Character.
Example:	The value `COUP` in the discount_type_code column will be masked to `9ccff020641cc1a68770161075cf33`.

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	DISCOUNTS_BASE	DISCOUNT_TYPE_CODE	CHARHASH	SHA-512			
3							

CHECKRUT

Social security numbers in Chile (RUT) follow a special format with a check digit at the end which is dependent on the first 8 digits of the number. CHECKRUT must have a Parm1 value – this is the name of another column in the same table which contains 2 or more RUT numbers.

Required parameters:	Parm1.
Applies to database types:	Numeric.

DOB

Adjust a date of birth by plus or minus Parm1. The age will NOT be adjusted relative to the current date or the override CDATE in the options file.

Required parameters:	Parm1.
Applies to database types:	Date.
Example:	The age 52 (DOB 9/5/1958) will remain at 52 but could be adjusted by, for example, between 10 and 30 days to 16/5/1958.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PERSONS	DATE_OF_BIRTH	DOB	10	30			N
3								

DOD

Adjust a date of death by plus or minus Parm1. The number of years since death will NOT be adjusted relative to the current date or override the CDATE in the options file.

Required parameters:	Parm1.
Applies to database types:	Date.
Example:	The date of death 10/9/2009 will remain at 1 year but could be adjusted by, between Parm1 and Parm2 to, for example, 7/9/2009.

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	PERSONS	DATE_OF_DEATH	DOD	10	30		
3							

DECRYPT

Creates a decrypted version of a column based on the key in Parm1.

Required parameters:	Parm1.
Applies to database types:	Character.
Example:	`e34;;=` could be converted to `ABC`.

DELETE

Deletes ALL data for the table, optionally using SQL in Parm1.

Required parameters:	Parm1 (Optional).
Applies to database types:	Character, Date and Numeric.
Example:	All data in table orders will be deleted.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	ORDERS	ORDER_ID	DELETE					N
3								

EMAIL

Masks the column with an auto-generated e-mail ID.

Required parameters:	None.
Applies to database types:	Character.
Example:	Column named email will be masked by auto-email IS.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	CARDPHONE	EMAIL	EMAIL					N
3								

ENCRYPT

Creates an encrypted version of a column based on the key in Parm1.

Required parameters:	Parm1.
Applies to database types:	Character.
Example:	`ABC` could be converted to `e34;;=`.

Note: The encrypted version will be longer than the original value, so you need to ensure the column width can accommodate the new value.

FIXED

Masks the column values with the fixed value provided in Parm1.

Required parameters:	Parm1. This is the fixed value.
Applies to database types:	Character, Date and Numeric.
Example:	Column ACCOUNT_NUMBER will be masked with the fixed value 11100022 and column PERSON_TYPE_CODE will be masked with the value CUST.

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	PAYMENT_OPTIONS	ACCOUNT_NUMBER	FIXED	111000222			
3	PAYMENT_OPTIONS	PERSON_TYPE_CODE	FIXED	CUST			
4							

Note: If you wish to set the value to null, enter the string `<NULL> - FIXED,<NULL>`. If you wish to maintain a space, enter the string `<SPACE> - FIXED,<SPACE>`.

FORMATMASK

Masks a character value - only changing lowercase, uppercase and numeric characters and leaving all others in place, i.e. maintaining the original format. It also guarantees uniqueness, and so can be applied to key columns.

Required parameters:	Parm1. This is the mask key. The mask key needs to be a series of alphanumeric characters - for example `AQgt76Wr`.
Applies to database types:	Alphanumeric.
Example:	Entering the mask key `K123abc-/345` will produce a masked output e.g. `L543dvs-/201`

GENCARD, MASTERCARD, VISACARD

Masks the column values with the 15 and 16 digit CREDIT CARD Numbers. You can also specify the card type you'd like to use (i.e. American Express, Mastercard, Credit Card).

Note: AMEX card numbers are 15 digits long.

Required parameters: None.

Applies to database types: Character and numeric.

Example:

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	CREDIT_CARDS	CARD_NUMBER	GENCARD					N
3	CREDIT_CARDS	CARD_NUMBER	AMEX					N
4	CREDIT_CARDS	CARD_NUMBER	MASTERCARD					N
5	CREDIT_CARDS	CARD_NUMBER	CARD					N
6								
7								

GUID

Generates a globally unique identifier - this will be a 36 character value.

Required parameters:	None.
Applies to database types:	Character.
Example:	

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PERSONS	LAST_NAME	GUID					
3								

HASH

Returns HASH Values for the integer fields.

Required parameters:	Parm1, Parm2 (Optional). Parm1 is the seed value for the hash and Parm2 gives the maximum value allowed.
Applies to database types:	Integer.
Example:	Here numbers column of table TEST1 have been hashed with the seed value of 35 and maximum allowed value of up to 5 digits.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	TEST1	NUMBERS	HASH	35	5			
3								

HASHPHONE

Maintains the first 4 digits of a phone number and hashes the rest based on a fixed key.

Required parameters:	None.
Applies to database types:	Number.
Example:	

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	PEOPLE	PHONE	HASHPHONE4				
3							

HASHRUT

Takes an existing RUT number (Chilean Social Security number) and hashes the first 8 digits, then adds the appropriate check digit to the end. This is the method that should be used to guarantee consistency across tables.

Note: The string length of a RUT is 9, so HASHRUT only works on string columns.

Required parameters:	None.
Applies to database types:	Numeric.
Example:	

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PEOPLE	RUT_NUMBER	HASHRUT					
3								

INTRANGE

Masks the column with values between Parm1 and Parm2.

Required parameters:	Parm1 and Parm2. Parm1 contains the start integer value of the range. Parm2 contains the end integer value of the range. The maximum value for Parm2 is 2147483647. If the database accepts decimal values, you can also use **NUMERICRANGE**.
Applies to database types:	Integer.
Example:	Column integer will be replaced by values between 100 and 110 and column integer_*nn* will be replaced by values between 120 and 200.

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	TRACK_MONTHLY	NUM_CARDS	INTRANGE	100	110		
3	TRACK_MONTHLY	NUM_TRANS	INTRANGE	10	15		
4							

IGNORE

Ignores the mask and retains value, if no cross-reference or default value can be found.

Optional parameters:	Parm1 and Parm2. Parm1 contains the cross-reference value. Parm2 contains the default value.
Example:	If Parm1 is absent, IGNORE will revert to using the default value set in Parm2. If Parm2 is also absent, IG ignore the mask and use existing value.

	A	B	C	D	E	F	
1	Table	Column	Function	Parm1	Parm2	Parm3	Pa
2	PERSONS	FIRST_NAME	IGNORE				
3							

NEXTVAL

Finds the next value from an Oracle sequence. If it is the first time the sequence has been used, it will start at 1, and then is incremented by 1 each subsequent sequence.

Required parameters:	Parm1 (name of the sequence). **Note:** You must have an XREF connection (must be Oracle) set if you use NEXTVAL.
Example:	If you called an Oracle sequence 'FirstSequence' and used it to update 20,000 fields in one run, the next time it was called, the run would start from 20,001.

NUMERICRANGE

Masks the column with numeric values between Parm1 and Parm2.

Required parameters:	Parm1 and Parm2. Parm1 contains the start value of the range. Parm2 contains the end value of the range.
Applies to database types:	Numeric.
Example:	Column float will be replaced by numeric values between 33.01 and 33.99.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	ORDERS	ORDER_TOTAL	NUMERICRANGE	40.01	49.99			N
3								

NUMHASH

Hashes a numeric value in a character column as digits.

Required parameters:	Parm1 (Parm2 and Parm3 are optional). Parm2 is the maximum length of the data. Parm3 is the minimum length of the data.
Applies to database types:	Character.
Example:	Numeric value in a character column will be hashed by the seed value to create a numeric string, of length between the values in Parm2 and Parm3.

OR

The `OR` function, in conjunction with `WHERE`, allows you to restrict your obfuscation to only certain rows.

For example, you can use separate masking rules from credit cards/direct debit expiration dates to invoices based on the `PAYMENT_TYPE_CODE` column. The `ADD` function must be used on a separate row to the `WHERE` function.

Note: The OR function is for masking flat files only as there is no SQL file.

Required parameters:	Parm1. This is an SQL `WHERE` clause.
Example:	

	A	B	C	D	E
1	Table	Column	Function	Parm1	Parm2
2	PERSONS		WHERE	PAYMENT_TYPE_CODE LIKE 'CC'	
3	PERSONS		OR	PAYMENT_TYPE_CODE LIKE 'DD'	
4	PERSONS	EXPIRE_DATE	ADDRANDOMDAYS	-7	21
5					

POSITIONMASK

Masks a value based on positional rules, as defined in Parm1.

Note: The OR function is for masking flat files only as there is no SQL file.

Required parameters:	Parm1 (Rules for each position are built from the following formulas): ■ RDXXXL – Random Digit at Position XXX from left ■ RDXXXR – Random Digit at Position XXX from right ■ RAXXXL – Random Alphabet at Position XXX from left ■ RAXXXR – Random Alphabet at Position XXX from right ■ RCXXXL - Random Alphanumeric character at position XXX from left ■ RCXXXR - Random Alphanumeric character at position XXX from right ■ F1XXXL - Fixed Digit 1 (for example) at Position XXX from left ■ F1XXXR - Fixed Digit 1 (for example) at Position XXX from right ■ FEXXXL - Fixed Alphabet E (for example) at Position XXX from left ■ FEXXXR - Fixed Alphabet E (for example) at Position XXX from right **Note:** If a rule is not given for any position, then the old value is retained post-masking. **Note:** If the old value is null or all blanks, skip the row and move on to the next row.
Example:	Column PHONE_NUMBER in Table PEOPLE will have the first 3 characters masked with the fixed value 9, with remaining digits remaining as original values. Therefore, the resulting masked value will be 999XXXXXX, where X existing value

	A	B	C	D	E
1	Table	Column	Function	Parm1	Parm2
2	PEOPLE	PHONE_NUMBER	POSITIONMASK	F9001L-F9002L-F9003L	
3					

Note: All of the rules for a column must be set in one row of the CSV, as above. To seperate each positional rule, p use a hyphen ('-') without a space.

RANDLOV

Masks the column values with the randomly selected values from the seed file.

Required parameters:	Parm1, the seed file name.
Applies to database types:	Character.
Example:	The data for RANDLOV functions can also be drawn from database tables. Select the columns you wish to use and place them in order in the mapping file (i.e. RD_REF_VALUE2, RD_REF_VALUE3 etc).

Note: Colums drawn from the database tables DO NOT contain a .txt suffix.

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	ADDRESS	CITY	RANDLOV	uktowns.txt			
3	ADDRESS	STATE_PROVINCE	RANDLOV	ukpostcode3.txt			
4	PERSONS	LAST_NAME	RANDLOV	lastnameindian.txt			
5							

RANDLOV1

Generates random addresses, cities, states/provinces etc., from a seed table, that are valid for an existing zip code.

Required parameters:	Parm1 (Seed File Name), Parm 2 (position of column in gtrsc_reference_ data) and Parm 3 (the column in code from). If you have a table like in the example below with PARM3 being the column POSTAL_CODE, which is use 3, 5, and 4 from the gtrsc_reference data seed table using the rd_ref_id of "US_ADDRESSES" and the pos
Optional parameters:	Parm4. This is a default value, which can be used if the rd_ref_value defined in Parm3 can not be found, o mask.
Example:	The data for RANDLOV functions can also be drawn from database tables. Select the columns you wish to the mapping file (i.e. RD_REF_VALUE2, RD_REF_VALUE3 etc). **Note:** Colums drawn from the database tables DO NOT contain a .txt suffix.

	A	B	C	D	E	
1	Table	Column	Function	Parm1	Parm2	Parm
2	ADDRESSES	ADDRESS2	RANDLOV1	US_ADDRESS	3	POST
3	ADDRESSES	CITY	RANDLOV1	US_ADDRESS	5	POST
4	ADDRESSES	STATE_PROVINCE	RANDLOV1	US_ADDRESS	4	POST
5						

RANDOM

Masks the column with random values between Parm1 and Parm2.

Required parameters:	Parm1 and Parm2. Parm1 contains the start value of the range. Parm2 contains the end value of the range. **Note:** Parm1 and Parm2 must be provided in the format YYYYMMDD if applied to a date.
Applies to database types:	Numeric and Date.
Example:	Column ORDER_DATE will be replaced by random date between 2001-Sept-18 to 2002-Nov-15.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	ORDERS	ORDER_DATE	RANDOM	20010918	20021115			N
3								

RANDOMDATE

Replaces existing value with a random value between Parm1 and Parm2.

Required parameters:	Parm1 and Parm2.
Applies to database types:	Character.
Example:	The following columns will have their values replaced by random date values between Parm1 and Parm2.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	ORDERS	ORDER_DATE	RANDOM	20010918	20021115			N
3								

RANDOMTXT

Replaces the column with random text.

Required parameters:	Parm1 and Parm2. Parm1: The minimum length of the text. Parm2: The maximum length of the text.
Applies to database types:	Character data types.
Example:	`RANDOMTXT,3,12`. Column will have the value `XrzFF`. The length of the text string will be between 3 and 12 long.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PERSONS	FIRST_NAME	RANDOMTXT	3	12			N
3								

RANDSSN

Masks the column with random values of social security number.

Required parameters:	Parm1 (Optional, which acts as separator for SSN).
Applies to database types:	Numeric and character types.
Example:	Columns listed in 'Column' will be replaced by a Random Social Security number. Entering a separator character into Parm1 (i.e. *), will generate a social security number like 987*65*4320.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PERSONS	ID	RANDSSN	*				N
3								

REPLACE

Searches the column values for the character pattern mentioned in Parm1 and replaces it with the character pattern mentioned in Parm2. Replace operation is case sensitive.

If only Parm1 is supplied then the value of the Parm1 should be a name of the CSV file that contains the list of values to be replaced. This CSV file should be placed in the same directory as `gtsdm.exe`.

<table>
<tr><td>Required parameters:</td><td>Parm1 and Parm2.

Parm1 contains the character pattern to be searched in the Column. If Parm2 is absent then only Parm1 is the name of the CSV file that contains list of values to be replaced.

Parm2 contains the character pattern to be replaced.</td></tr>
<tr><td>Applies to database types:</td><td>Character</td></tr>
<tr><td>Example:</td><td>Case 1 : When Parm2 is present.

Here pattern ‘<code>Ab</code>’ when found in the column ‘char’ is replaced by ‘<code>23</code>’, ‘<code>a</code>’ if found in the column ‘char_nn’ is also replaced by ‘<code>23</code>’.
<table>
<tr><td></td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td><td>G</td></tr>
<tr><td>1</td><td>Table</td><td>Column</td><td>Function</td><td>Parm1</td><td>Parm2</td><td>Parm3</td><td>Parm4</td></tr>
<tr><td>2</td><td>PERSONS</td><td>ADDRESS_1</td><td>REPLACE</td><td>Ab</td><td>23</td><td></td><td></td></tr>
<tr><td>3</td><td>PERSONS</td><td>ADDRESS_2</td><td>REPLACE</td><td>a</td><td>23</td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
Case 2 : When Parm2 is absent.

Here Parm2 is absent and Parm1 is the name of the CSV file that contains a list of values to be replaced.
<table>
<tr><td></td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td></tr>
<tr><td>1</td><td>Table</td><td>Column</td><td>Function</td><td>Parm1</td><td>Parm2</td><td>Parm3</td></tr>
<tr><td>2</td><td>PERSONS</td><td>ADDRESS_1</td><td>REPLACE</td><td>replacelist.csv</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
The example CSV file could look like shown below. Also please note that <code>REPLACE.csv</code> is placed in the same directory as <code>GTSDM.exe</code></td></tr>
</table>

	Replace.csv - Notepad File Edit Format View Te, AD st, CD ing, EH 12, 00 G, g

RJUST

RJUST strips blanks from the right of the string and right justifies the column in the string, padding the left with blanks (default) or the value defined in Parm1.

Required parameters:	Parm1 (Optional).
Applies to database types:	Character.
Example:	The example would mask the LAST_NAME (e.g. Smith) as something like “55555SMITH”. **Note:** This function is most likely to be used in conjunction with `substr` functionality. The string is right justified to the size of the column (if `substr` is being masked, the column size is assumed to be the end of the substring).

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PEOPLE	LAST_NAME	RJUST	5				N
3								

RUT

Generate a Chilean Social Security Number.

Required parameters:	None.
Applies to database types:	Character.

Example:

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	PERSONS	SOCIAL_SECURITY_NUMBER	RUT				
3							

SEQCHAR

A sequential character using BASE62 numbers, for example, starting at AAA will give AAA then AAB etc.

Required parameters:	Parm1.
Applies to database types:	Character.
Example:	

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PERSONS	CONFIRMED_EMAIL	SEQCHAR	AAA				N
3								

SEQLOV

Masks the column values with the sequentially selected values from the seed file.

Required parameters:	Parm1. The seed file name.
Applies to database types:	Character types.
Example:	The data for SEQLOV functions can also be drawn from database tables. Select the columns you wish to use and place them in order in the mapping file (i.e. RD_REF_VALUE2, RD_REF_VALUE3 etc). **Note:** Colums drawn from the database tables DO NOT contain a .txt suffix.

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	ADDRESS	CITY	SEQLOV	uktowns.txt			
3	ADDRESS	STATE_PROVINCE	SEQLOV	ukpostcode3.txt			
4	PERSONS	LAST_NAME	SEQLOV	lastnameindian.txt			

SEQLOV1

Generates sequential addresses, cities, states/provinces etc.,from a seed table, that are valid for an existing zip code.

Required parameters:	Parm1 (Seed File Name), Parm 2 (position of column in gtrsc_reference_ data) and Parm 3 (the column in code from). If you have a table like in the example below with PARM3 being the column POSTAL_CODE, which is us 3, 5, and 4 from the gtrsc_reference data seed table using the rd_ref_id of "US_ADDRESSES" and the pos
Optional parameters:	Parm4. This is a default value, which can be used if the rd_ref_value defined in Parm3 can not be found, c mask.
Example:	

	A	B	C	D	E	
1	Table	Column	Function	Parm1	Parm2	Pa
2	ADDRESSES	ADDRESS2	SEQLOV1	US_ADDRESSES	3	PC
3	ADDRESSES	CITY	SEQLOV1	US_ADDRESSES	5	PC
4	ADDRESSES	STATE_PROVINCE	SEQLOV1	US_ADDRESSES	4	PC
5						

SEQNUMBER

Updates each row with a user defined sequence.

For example, if Parm1 is 10 the first row is updated for this column with value 10, the next row with 11 etc. If Parm1 is not supplied the sequence starts at 1.

Optional parameters:	Parm1. The start value for the sequence.
Example:	

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PERSONS	ID	SEQNUMBER	10				N
3								

SHUFFLE

Shuffles the values in the specified column for an entire table. The function creates a seed file by writing out each row's value for the column to a list of values.

It then performs a **RANDLOV** of **SEQLOV** function to overwrite the values in the database.

The Shuffle function allows you to write this list of values to:

1. File - SDM knows to write to file if a . is present in the parm1 value.
2. Database - If the parm1 value does not have a . in it, the category name is stored in the database seed table.

Required parameters:	Parm1. This is the category in the seedtable in which your list of values is saved, for example, MY_ADDRESSES containing Address, City and Postcode. Parm2. This is the column in the seedtable you wish to place the value in, for example, entering 1 would select RD_REF_1, which in this example is Address.
Required Options:	In order for the function to run, you will need to use the following (For more information on this, please go to the **Options File section** on the manual): ■ SEEDTABLECONNECT=connectscramble.txt ■ SEEDTABLE=gtsrc_reference_data ■ SEEDTABLECOLUMNS=RD_REF_ID, RD_REF_VALUE1, RD_REF_VALUE2 etc
Applies to database types:	All types.
Example:	**Note:** 1. Do not use the name of an existing seed table.txt file as this will be overwritten each time it is run. 2. This function can only be used on databases not flat files.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	ADDRESSES	ADDRESS_1	SHUFFLE	MY_ADDRESS	1			N
3	ADDRESSES	CITY	SHUFFLE	MY_ADDRESS	2			
4	ADDRESSES	POSTCODE	SHUFFLE	MY_ADDRESS	3			

SQLFUNCTION

Allows you to make a callout to a native database function or a user defined function. You can also use this to use normal SQL operators to process combinations of other columns. All SQL functions with the exception of aggregate functions should work.

Required parameters:	Parm1, the SQL function or SQL statement.
Applies to database types:	All types.
Example:	`Parm1=first_name || ' '|| last_name`. This will concatenate first name, a space, and a last name. `Parm1=mynumberformat(HHNO)`. This will pass HHNO into the database function. The returned value will populate the masked column.

TIN

Generates a US (United States) Tax Identification Number.

Required parameters:	None.
Applies to database types:	Character and numeric types.
Example:	Column TAXID in table PEOPLE will be masked with a generated US Tax Identification Number.

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm
2	PEOPLE	TAXID	TIN				
3							

TRANSLATE

Searches the column values for every single character mentioned in Parm1 and replaces it with the sequentially corresponding character mentioned in Parm2. TRANSLATE is a character-by-character operation.

Required parameters:	Parm1 and Parm2. Parm1 contains the character(s) to be searched in the Column. Parm2 contains the corresponding character(s) to be replaced.
Applies to database types:	Character types.
Example:	All instances of 'a' in column FIRST_NAME will be translated to 'x', and all instances of 1 in column MEMBERSHIP_ID will be translated to 6.

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	PERSONS	FIRST_NAME	TRANSLATE	a	x		
3	PERSONS	MEMBERSHIP_ID	TRANSLATE	1	6		
4	PERSONS	PERSON_TYPE_CODE	TRANSLATE	st	kw		

TRANSPOSE

Converts characters consistently to other characters. A to C, B to D etc, set Parm1 to a number to act as a key.

Required parameters:	Parm1. It contains the key of the transposition.
Applies to database types:	Character types.
Example:	'ab' if found in the column value will be translated to 'ef' i.e. every 'a' character translated to 'e' and every 'b' character translated to 'f'. This will vary depending on the key.

TRUNCATE

Truncates ALL data for the table. The TRUNCATE function can also be used with a WHERE clause.

Required parameters:	None.
Applies to database types:	Character, Date and Numeric.
Example:	Truncating the data will execute a fast delete of ALL the data in the table.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PERSONS	LAST_NAME	TRUNCATE					N
3								

USPHONE

Masks the column with an auto-generated 7digit US Phone number of the format `xxx-xxxx`.

Required parameters:	None.
Applies to database types:	Character types.
Example:	Column named usphone will be masked by auto-generated 7-digit US Phone numbers.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PERSONS	USPHONE	USPHONE					
3								

USPHONE(10)

Masks the column with an auto-generated 10 digit US Phone number of the format `xxx-xxx-xxxx`.

Required parameters:	None.
Applies to database types:	Character types.
Example:	Column named usphone10 will be masked by auto-generated 10-digit US Phone numbers.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PERSONS	USPHONE	USPHONE(10)					
3								

USZIP

Masks the column with an auto-generated 5-digit US Zip code.

Required parameters:	None.
Applies to database types:	Character types.
Example:	Column named uszip will be masked by auto-generated 5-digit US Zip code.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PERSONS	ZIP_CODE	USZIP					
3								

USZIP+4

Masks the column with an auto-generated 9-digit US Zip+4 code of the format `xxxxxxxxx`.

Required parameters:	None.
Applies to database types:	Character types.
Example:	Column named uszip will be masked by auto-generated 9-digit US Zip code.

	A	B	C	D	E	F	G	H
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4	KeepNulls
2	PERSONS	ZIP_CODE	USZIP+4					
3								

VALIDSSN

Identifies whether a column contains a valid SSN (United States Social Security Number), then, if yes, masks with a generated SSN.

Required parameters:	None.
Applies to database types:	Character and numeric types.
Example:	Column ID in table PEOPLE will be replaced by a Random Social Security number, if a valid Social Secur Number is found in the column.

	A	B	C	D	E	F
1	Table	Column	Function	Parm1	Parm2	Parm3
2	PEOPLE	ID	VALIDSSN			
3						

VALIDSSNSUB

Identifies whether the first 9 characters of a column contains a valid SSN (United States Social Security Number), then, if yes, masks with a generated SSN.

Required parameters:	None.
Applies to database types:	Character and numeric types.
Example:	Column ID in table PEOPLE will be replaced by a Random Social Security number, if a valid Social Security Num is found in the first 9 characters of the column.

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	PEOPLE	ID	VALIDSSNSUB				
3							

VALIDTIN

Identifies whether the column contains a valid TIN (United States Tax Identification Number), then, if yes, masks with a generated TIN.

Required parameters:	None.
Applies to database types:	Character and numeric types.
Example:	Column ID in table PEOPLE will be replaced by a Random Tax Identification Number, if a valid TIN is found in the column.

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	PEOPLE	TAXID	VALIDTIN				
3							

VARIANCE

Variant values are generated based on Parm1 (% value) and then added to or subtracted from the column values.

Required parameters:	Parm1, Parm2(Optional), Parm3(Optional). Parm1 gives percentage variance (1 - 99). Parm2 is the minimum permitted value. Parm3 is the maximum permitted value.
Applies to database types:	Numeric types.
Example:	If the column value is 100 then Parm1 applies 60% variance and a random number is generated between 40 and 160. However Parm2 (minimum permitted value) and Parm3 (maximum permitted value) ensures that the generated random value lies in the range 50 - 150 instead of 40 – 160.

	A	B	C	D	E	F	G
1	Table	Column	Function	Parm1	Parm2	Parm3	Parm4
2	PERSONS	CREDIT_SCORE	VARIANCE	60	50	150	
3							

WHERE

The WHERE function allows you to restrict your obfuscation to only certain rows in the table. This will allow you to mask, for example, Male names and Female names differently based on the GENDER column.

The WHERE function does not require a column to be entered. Parm1 will contain the SQL WHERE clause used to sub-select the table data.

Required parameters:	Parm1. Parm1 is an SQL WHERE clause. It is worth running the WHERE clause in a standard SQL IDE to validate the syntax is correct.
Applies to:	Entire table.
Example:	If the MSP_CODE is like HSD **Note:** SDM currently supports the following operators: <, <=, =, >=, >, and LIKE (LIKE is not supported for flat files. It is only supported for database masking).

	A	B	C	D	E
1	Table	Column	Function	Parm1	Parm2
2	QMS_MESSAGE_TEXT	HELP TEXT	RANDLOV	companies.txt	
3	QMS_MESSAGE_TEXT		WHERE	MSP_CODE_LIKE 'HSD%'	
4	QMS_MESSAGE_TEXT	CREATED BY	FIXED	huw	
5	QMS_MESSAGE_TEXT		WHERE	MSP_CODE_LIKE 'OFG%'	
6	QMS_MESSAGE_TEXT	CREATED BY	FIXED	fred	
7	QMS_MESSAGE_TEXT	MESSAGE_TEXT	RANDLOV	lastnames.txt	
8					

If you wish to issue multiple WHERE clauses, repeat the WHERE function. Each WHERE function will refer to the block of masking functions BELOW it in the CSV.

In the above example, all of the columns HELP_TEXT will be masked with random values from companies.txt; the rows with an MSP_CODE starting with 'HSD' will have the value huw assigned to CREATED_BY; and the rows with an MSP_CODE starting with 'OFG' will have the value fred assigned to CREATED_BY and a random value from the lastnames.txt file assigned to column MESSAGE_TEXT.

17 Managing Primary Keys and Foreign Keys

If you wish to mask primary keys, unique indexes or any matching columns using foreign keys you may have to drop these before masking and reapply them after masking. Ask your DBA to provide a script or use Datamaker to generate any scripts you need.

When you do mask primary keys you need to be careful if you are likely to create duplicate values as you go. For example, if you have some ID values of 2,5,7,8,100 and you apply a SEQNUMBER masking function starting at 100 then the value 2 will be updated to a value of 100 which is a duplicate.

In addition, masking tables without a primary key can be slow as most databases try and perform a full table update. If you have any questions please contact support@softwareag.com before you begin the masking.

18 Managing Large Table Updates or Large Seed Tables

If you are masking very large tables, have a large number of seed tables or very large seed tables you will need to increase the memory allocated to the GTSDM executable.

There is a new command file in the install directory called `BIGSDM`, which may be needed when masking large data or seed tables, or many seed tables.

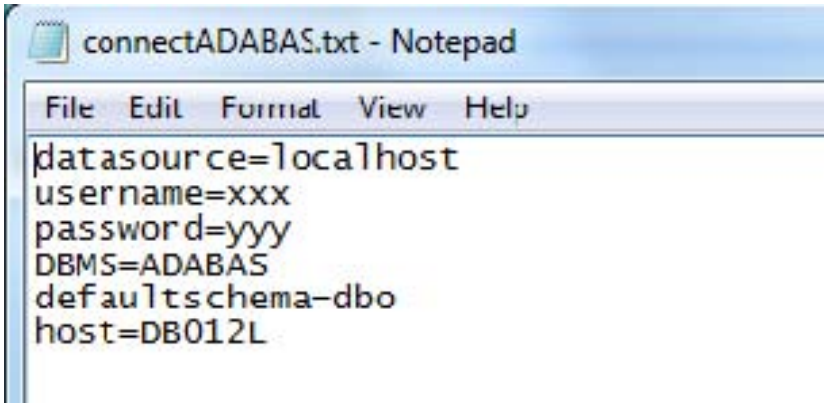
The syntax is

```
BIGSDM 1000 connect,.txt map.csv [options.txt]
```

Note: In the example above, 1000 is the virtual memory in Megabytes, or 1GB.

19 Appendix 1: Connection Files

Adabas


connectADABAS.txt - Notepad
File Edit Format View Help
datasource=localhost
username=xxx
password=yyy
DBMS=ADABAS
defaultschema-dbo
host=DB012L

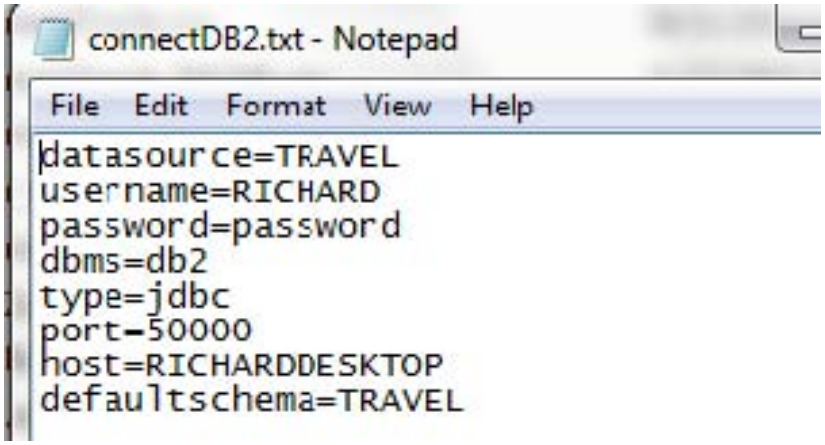
Cache

connectCache.txt - Notepad

File Edit Format View Help

```
datasource=Samples
dbms=Cache
username=sampleuser
password=dylan110
type=jdbc
port=1972
host=localhost
defaultschema=Cinema
```

db2


connectDB2.txt - Notepad
File Edit Format View Help
datasource=TRAVEL
username=RICHARD
password=password
dbms=db2
type=jdbc
port=50000
host=RICHARDDESKTOP
defaultschema=TRAVEL

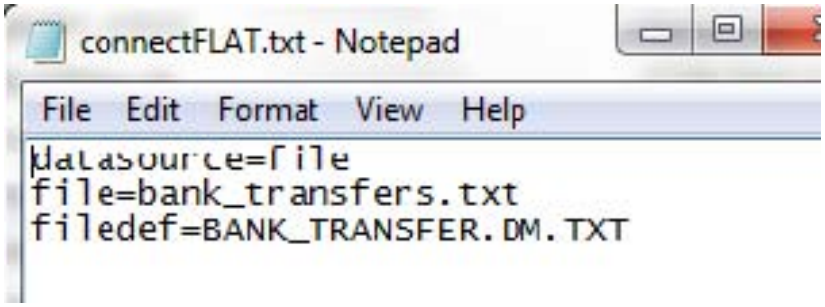
db2 400

connectdb2400.txt Notepad

File Edit Format View Help

```
datasource=Gridtest1
username=gridtools
password=freddy
dbms=db2400
type=jdbc
port=50000
host=iseriesmachine
defaultschema=Gridtest1
```

FLAT files



The datasource must be set to file.

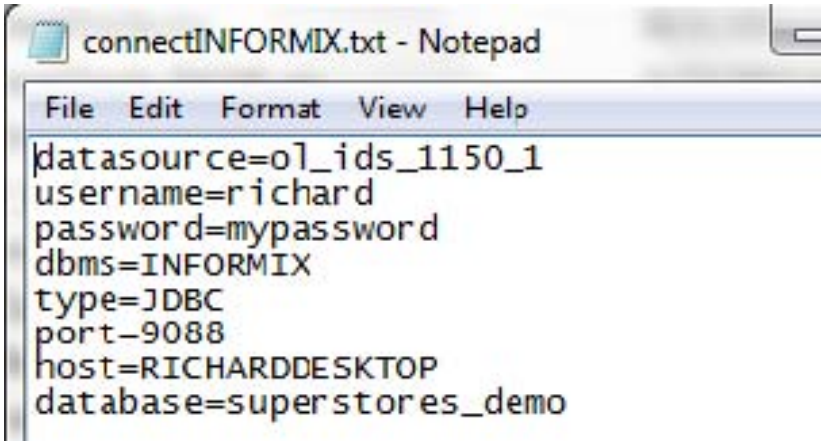
The file must point to the name of the file to be masked.

The filedef must point to a definition file containing the layout of the file to be masked.

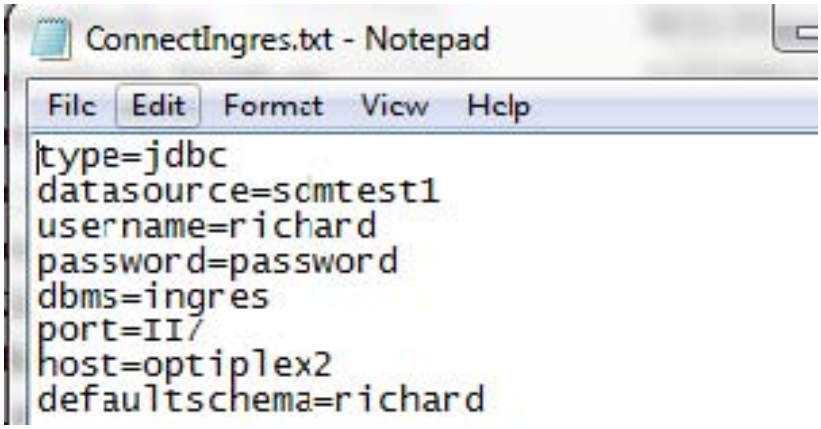
See the section ***Building Map Files for FLAT Files*** for details on how to build this file.

The masker will create an output file which masks the data. This file will add a suffix `.scramble` to the original name.

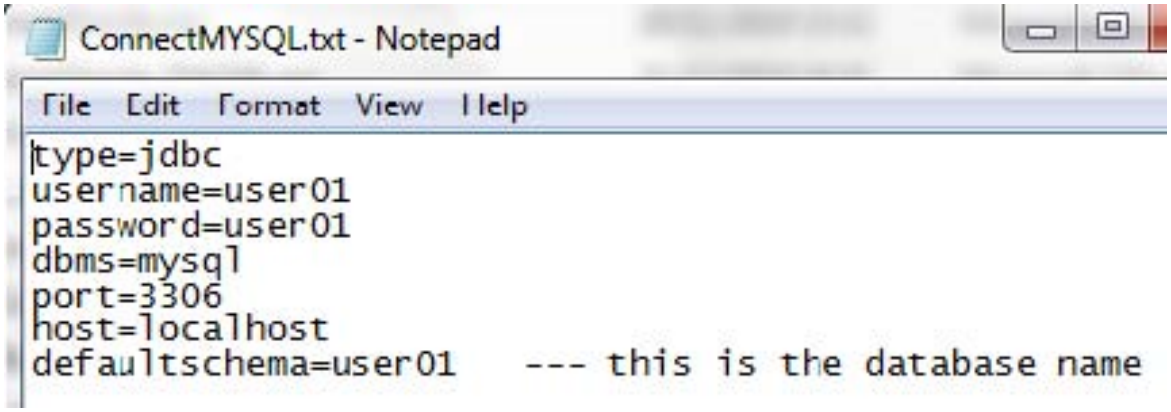
Informix


connectINFORMIX.txt - Notepad
File Edit Format View Help
datasource=ol_ids_1150_1
username=richard
password=mypassword
dbms=INFORMIX
type=JDBC
port=9088
host=RICHARDDESKTOP
database=superstores_demo

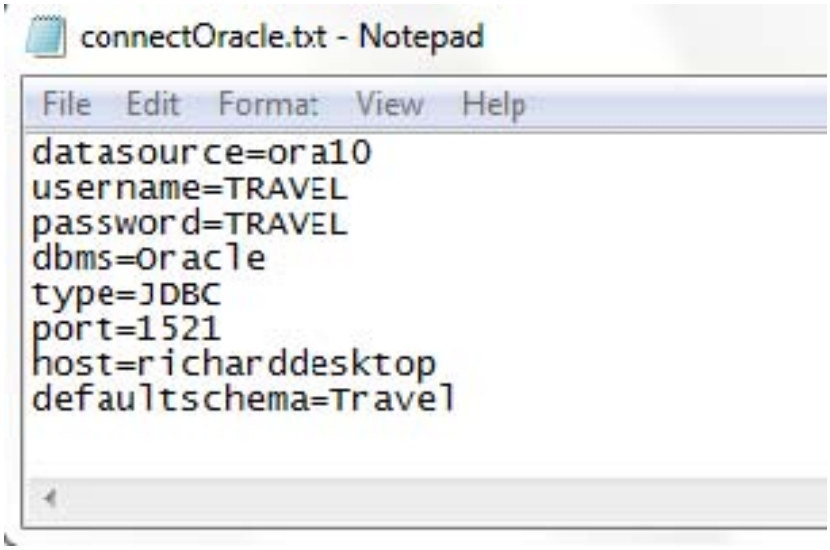
Ingres


ConnectIngres.txt - Notepad
File Edit Format View Help
type=jdbc
datasource=scmtest1
username=richard
password=password
dbms=ingres
port=II7
host=optiplex2
defaultschema=richard

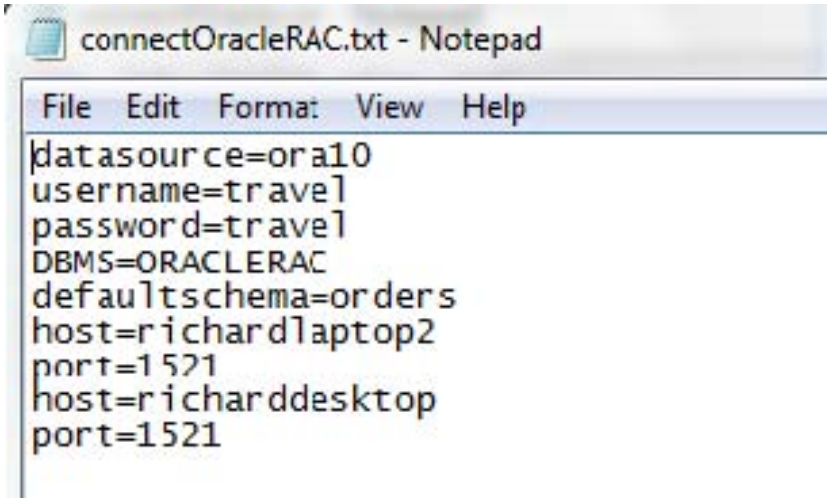
MySQL


ConnectMYSQL.txt - Notepad
File Edit Format View Help
type=jdbc
username=user01
password=user01
dbms=mysql
port=3306
host=localhost
defaultschema=user01 --- this is the database name

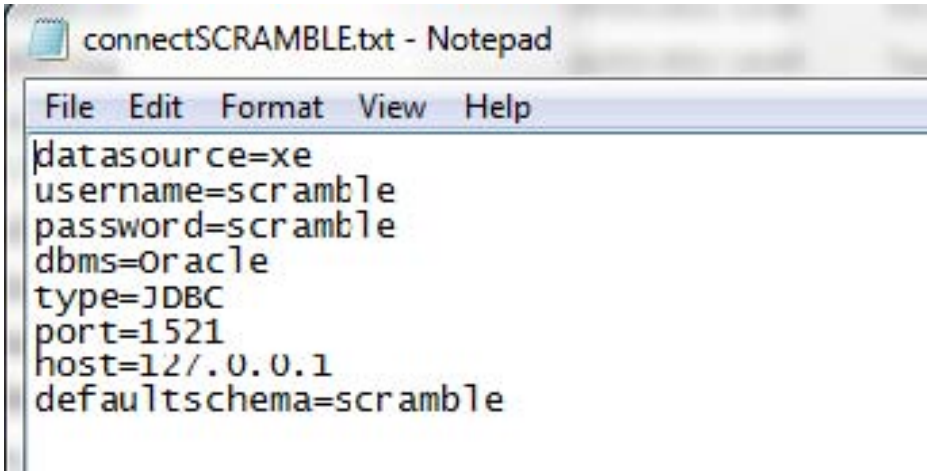
Oracle


connectOracle.txt - Notepad
File Edit Format View Help
datasource=ora10
username=TRAVEL
password=TRAVEL
dbms=Oracle
type=JDBC
port=1521
host=richarddesktop
defaultschema=Travel

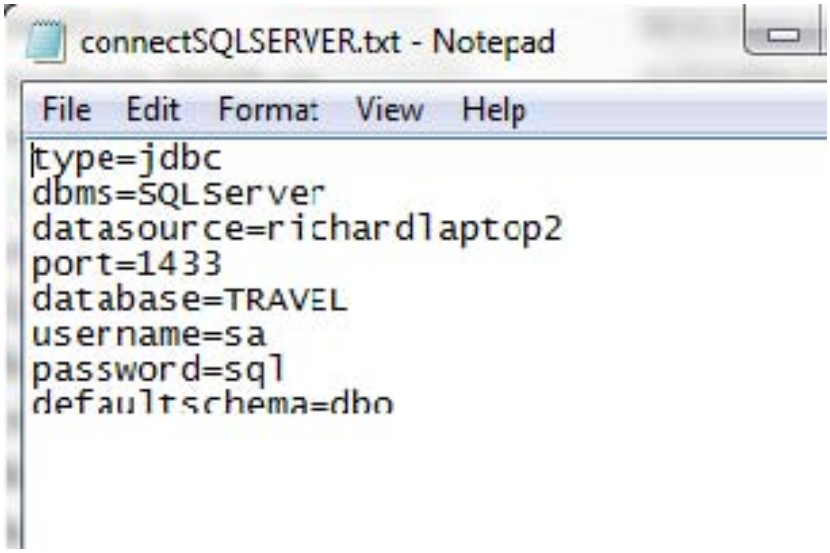
OracleRAC


connectOracleRAC.txt - Notepad
File Edit Format View Help
datasource=ora10
username=travel
password=travel
DBMS=ORACLERAC
defaultschema=orders
host=richardlaptop2
port=1521
host=richarddesktop
port=1521

Scramble


connectSCRAMBLE.txt - Notepad
File Edit Format View Help
datasource=xe
username=scramble
password=scramble
dbms=Oracle
type=JDBC
port=1521
host=127.0.0.1
defaultschema=scramble

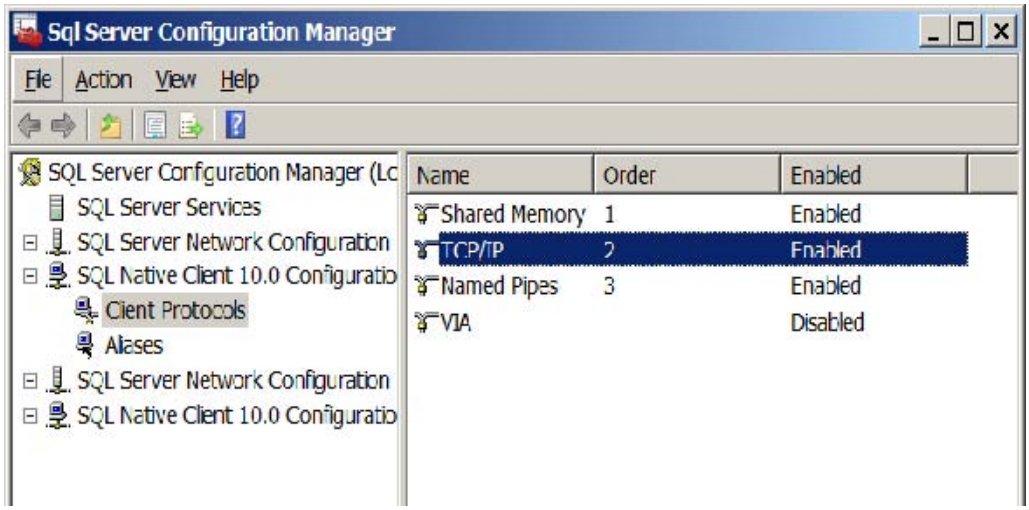
SQL Server

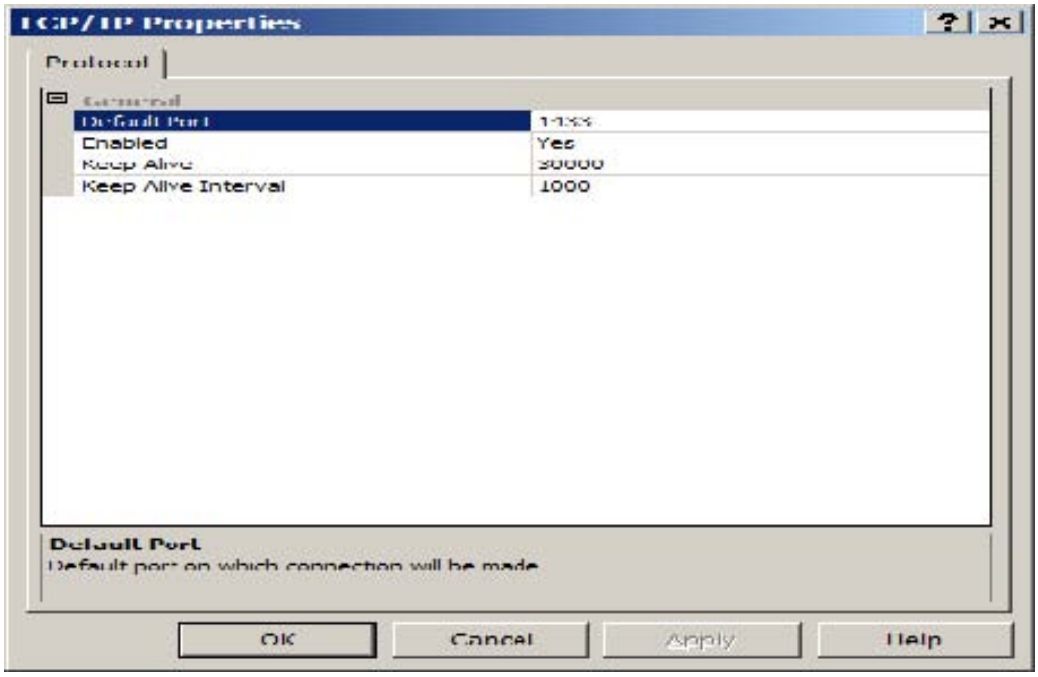


If you are unsure of the name of SQLServer machine and database, check with your DBA. The datasource value should be the IP address or name of the server your SQL server database is running on, not the SQL Server service name as shown below.

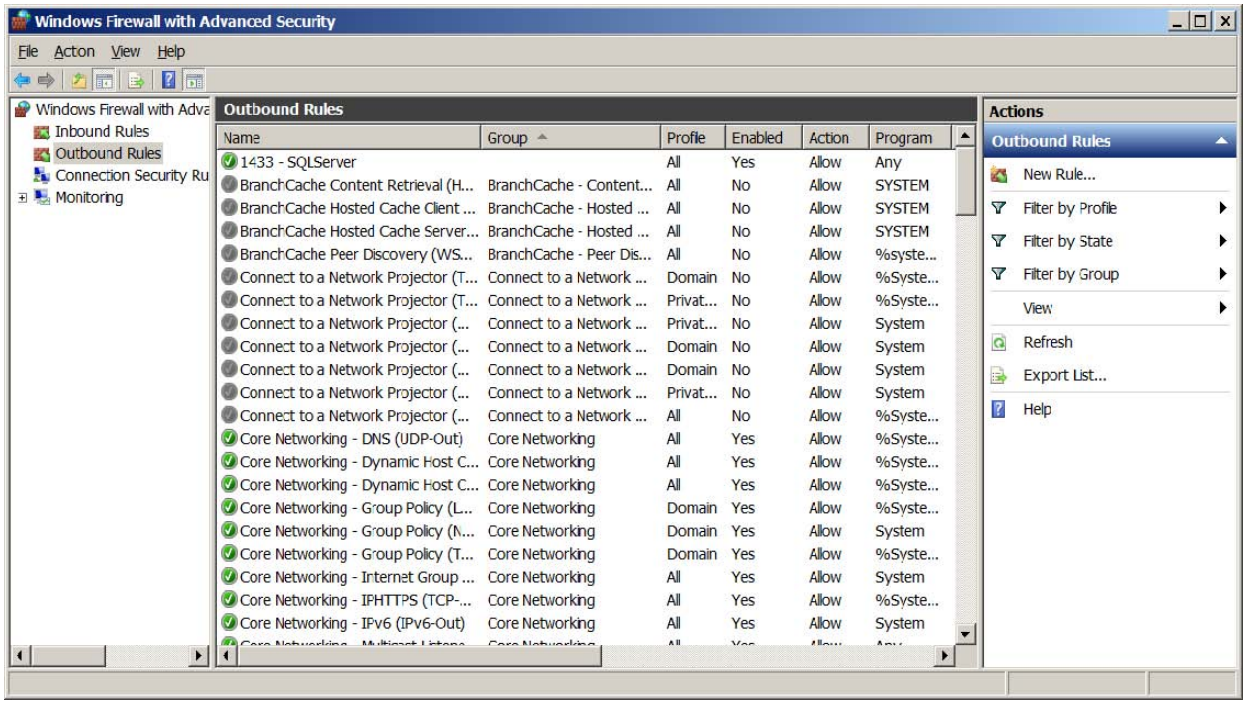
Before you connect to the database make sure you can connect using SQLServer Management studio. You must connect using native connectivity NOT native windows connectivity.

If you are unsure of the port, use configuration manager and check the TCP/IP properties.





In addition, you may need to include a firewall rule to allow remote access to the port. For example, set up inbound and outbound rules for Windows 7.

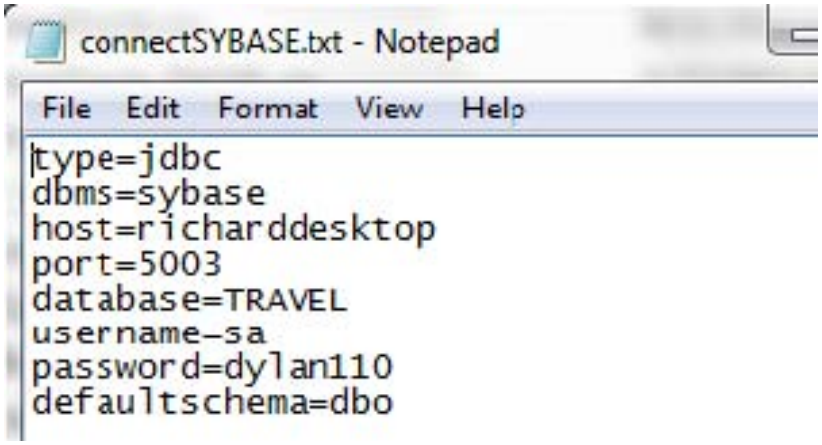

Windows Firewall with Advanced Security
File Action View Help
Windows Firewall with Adva
Inbound Rules
Outbound Rules
Connection Security Ru
Monitoring
Outbound Rules
Name Group Profile Enabled Action Program
1433 - SQLServer All Yes Allow Any
BranchCache Content Retrieval (H... BranchCache - Content... All No Allow SYSTEM
BranchCache Hosted Cache Client ... BranchCache - Hosted ... All No Allow SYSTEM
BranchCache Hosted Cache Server... BranchCache - Hosted ... All No Allow SYSTEM
BranchCache Peer Discovery (WS... BranchCache - Peer Dis... All No Allow %syste...
Connect to a Network Projector (T... Connect to a Network ... Domain No Allow %Syste...
Connect to a Network Projector (T... Connect to a Network ... Privat... No Allow %Syste...
Connect to a Network Projector (... Connect to a Network ... Privat... No Allow System
Connect to a Network Projector (... Connect to a Network ... Domain No Allow System
Connect to a Network Projector (... Connect to a Network ... Domain No Allow System
Connect to a Network Projector (... Connect to a Network ... Privat... No Allow System
Connect to a Network Projector (... Connect to a Network ... All No Allow %Syste...
Core Networking - DNS (UDP-Out) Core Networking All Yes Allow %Syste...
Core Networking - Dynamic Host C... Core Networking All Yes Allow %Syste...
Core Networking - Dynamic Host C... Core Networking All Yes Allow %Syste...
Core Networking - Group Policy (L... Core Networking Domain Yes Allow %Syste...
Core Networking - Group Policy (N... Core Networking Domain Yes Allow System
Core Networking - Group Policy (T... Core Networking Domain Yes Allow %Syste...
Core Networking - Internet Group ... Core Networking All Yes Allow System
Core Networking - IPHTTPS (TCP-... Core Networking All Yes Allow %Syste...
Core Networking - IPv6 (IPv6-Out) Core Networking All Yes Allow System
Actions
Outbound Rules
New Rule...
Filter by Profile
Filter by State
Filter by Group
View
Refresh
Export List...
Help

SQL Anywhere

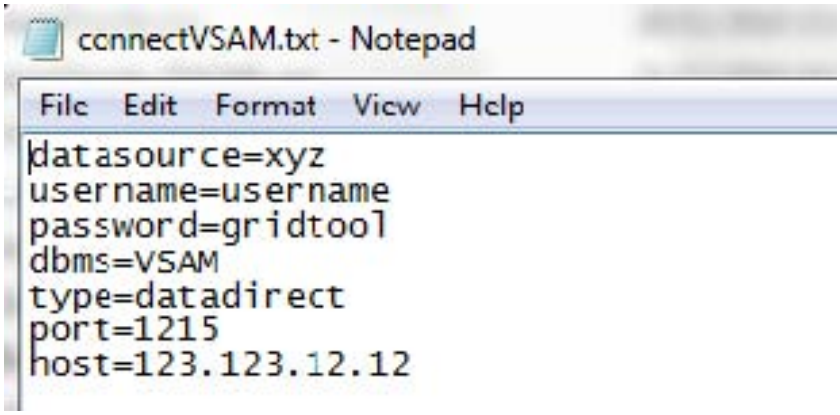
```
File  Edit  Format  View  Help
datasource=sql anywhere 10
database-course1 sql anywhere
username=dba
password=sql
dbms=sql anywhere
type=jdbc
defaultschema=dba


*****The datasource value is the ODBC driver name
*****The database value is the ODBC DSN name - this sh
```

Sybase

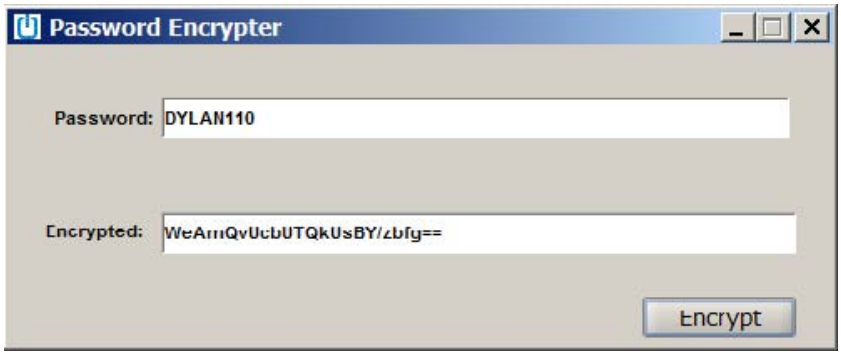

connectSYBASE.txt - Notepad
File Edit Format View Help
type=jdbc
dbms=sybase
host=richarddesktop
port=5003
database=TRAVEL
username=sa
password=dylan110
defaultschema=dbo

VSAM or SHADOW DIRECT



Managing Passwords

If you do not wish to have un-encrypted passwords in the connection text file, run the utility `gtencypter.exe`. This will present the following screen:



Type in the password and press Encrypt.

Copy the resulting string into the `connection.txt` file replacing the keyword `password=` with `epassword=`.