|  |
| --- |
| Capax Global |
| Capax Data Migration Toolkit Guide |
| Prepared for Credit Suisse |

|  |
| --- |
| Capax Global  5/12/2012 |

C:\Users\Jerry Hawk\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Outlook\1Z1JSF0M\CapaxGlobalLogo144dpi (2).jpg

Contents

[Capax Data Migration Toolkit 2](#_Toc321338401)

[Prerequisites 2](#_Toc321338402)

[Components 2](#_Toc321338403)

[Configuration 2](#_Toc321338404)

[Configuration Keys 3](#_Toc321338405)

[General Workflow 5](#_Toc321338406)

[Appendix A – Object Mapping Support 9](#_Toc321338407)

[Appendix B – Data Replacement Support 10](#_Toc321338408)

[Appendix C – Annoted DataMigration.SybtoSql\_CopyDataThread 11](#_Toc321338409)

[Appendix D – Object Listing 17](#_Toc321338410)

# Capax Data Migration Toolkit

The Capax Data Migration Toolkit (CDMT) is a facility for migrating data out of Sybase and into Microsoft SQL Server. It is composed primarily of a SQL Server database, supplemented by Sybase client tools.

## Prerequisites

* Sybase Client tools 12.5.4 or greater (note, a reboot may be required after installing Sybase Client tools).
* Microsoft SQL Server 2008 x64 (all editions, Express and up)

## Components

The CDMT is primarily a SQL Server database supplemented by the Sybase Client Tools. The external interface is composed of the items below. See appendix for a complete listing.

**CapaxMigrationToolkit Database**

* Schemas
  + DataMigration
  + UserMigration
* Tables
  + DataMigration.SybtoSql\_Config
  + DataMigration.SybtoSql\_GeneralConfig
  + DataMigration.SybtoSql\_ObjectsToCopy\_Destination
  + DataMigration.SybtoSql\_ObjectsToCopy\_Source
  + DataMigration.SybtoSql\_QueueDetail
  + DataMigration.SybtoSql\_UserStore
* Views
  + DataMigration.SybtoSql\_UserContext
* Stored Procedures
  + DataMigration.MapDestinationFromSource
  + DataMigration.PopulateQueue
  + DataMigration.SybtoSql\_CopyData
  + UserMigration.SybtoSql\_CopyLogins
  + UserMigration.SybtoSql\_CopyUsers

## Configuration

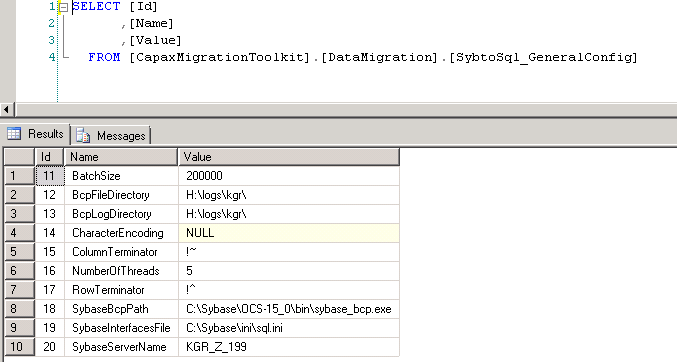
There are 5 tables dealing with configuration.

* DataMigration.SybtoSql\_Config  
  This table contains control data which is shared by the continuous copy system. This table does not have to be modified for the CDMT On-Demand which is the mode used for Entity Setup.
* DataMigration.SybtoSql\_GeneralConfig  
  This table contains name/value pairs representing the configuration parameters available. The current list includes:
* DataMigration.SybtoSql\_ObjectsToCopy\_Destination  
  This table contains the listing of objects into which data should be copied. Most often this listing will contain the same items in the source. See the section on source to destination mapping.
* DataMigration.SybtoSql\_ObjectsToCopy\_Source  
  This table contains the listing of objects from which data should be copied. Most often this listing will contain the same items in the destination. See the section on source to destination mapping.
* DataMigration.SybtoSql\_UserStore

DataMigration.SybtoSql\_UserStore (exposed by the view DataMigration.SybtoSql\_UserContext for writes) contains the user credentials to be used for the data migration (including logins and users).

### Configuration Keys

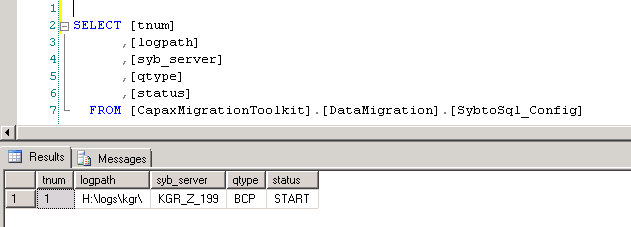
Basic configuration begins with data insertion into both DataMigration.SybtoSql\_GeneralConfig and DataMigration.SybtoSql\_Config. See screenshots below for keys and example values.



#### SybtoSql\_GeneralConfig

* BatchSize  
  The number of rows set in the batch size parameter of the bulk copy out of Sybase
* BcpFileDirectory  
  The directory where the BCP files are placed
* BcpLogDirectory  
  The directory where the BCP log files are placed
* CharacterEncoding  
  Reserved for future use
* ColumnTerminator  
  This character value is used as the column separator for the BCP and BULK INSERT commands
* NumberOfThreads  
  Reserved for future use. Multithreading is currently handled outside of CDMT via SQL Agent for example.
* RowTerminator  
  This character value is used as the row separator for the BCP and BULK INSERT commands
* SybaseBcpPath  
  The fully-qualified path name of the Sybase BCP executable
* SybaseInterfacesFile  
  The fully-qualified path of the Sybase interfaces file (sql.ini)
* SybaseServerName  
  The name of the ASE instance from which to copy data

#### SybtoSql\_Config



For CDMT On-demand mode, this table contains one row. The configuration values specified are superseded by those in SybtoSql\_GeneralConfig. For reference the explanation is provided below.

* tnum  
  Thread number – corresponds to an Agent job number. Set to 1 for On-demand mode.
* logpath  
  The directory where BCP and log files are kept
* syb\_Server  
  The name of the ASE instance from which to copy data
* qtype  
  Should be BCP for this installation
* status  
  Should be START for On-demand mode.

## General Workflow

#### Operation

Operation of CDMT can be best illustrated through example. The following sequence of calls illustrates typical configuration and usage of CDMT.

The first sequence of calls is made to set configuration.

insert into CapaxMigrationToolkit.DataMigration.SybtoSql\_UserStore values( N'GDX\_32\_12', N'kplus', N'sa', convert(varbinary(255),N'Password'), '', '', '', convert(varbinary(255),N''))

update CapaxMigrationToolkit.DataMigration.SybtoSql\_GeneralConfig set Value = N'c:\Sybase\OCS-15\_0\ini\sql.ini' where Name = 'SybaseInterfacesFile'

update CapaxMigrationToolkit.DataMigration.SybtoSql\_GeneralConfig set Value = N'FA\_33\_NR\_CDMT' where Name = 'SybaseServerName'

update CapaxMigrationToolkit.DataMigration.SybtoSql\_GeneralConfig set Value = N'F:\CapaxMigrationToolkit\' where Name = 'BcpFileDirectory'

update CapaxMigrationToolkit.DataMigration.SybtoSql\_GeneralConfig set Value = N'F:\CapaxMigrationToolkit\' where Name = 'BcpLogDirectory'

insert into CapaxMigrationToolkit.DataMigration.SybtoSql\_Config values( 1, N'F:\CapaxMigrationToolkit\', N'FA\_33\_NR\_CDMT', 'BCP', 'START')

The next set of calls is done to setup the data that is to be copied. As stated previously, CDMT can copy three types of “data”. The following lines demonstrate how to configure each.

* User table data
  + This can be specified by either setting the ObjectType to be a “Database” (short-hand for specifying all tables)
  + By specifying which tables are to be copied (using the full three part name – database.owner.table)
* Logins  
  Note, as logins are scoped at the instance level, the three-part source must be “master.dbo.syslogins”.
* Database users  
  Note, the Name property must be the three-part name

truncate table DataMigration.SybtoSql\_ObjectsToCopy\_Source

insert into DataMigration.SybtoSql\_ObjectsToCopy\_Source ( Name, ObjectType ) values ('ktpplus', 'Database' )

insert into DataMigration.SybtoSql\_ObjectsToCopy\_Source ( Name, ObjectType ) values ('ktparch', 'Database' )

insert into DataMigration.SybtoSql\_ObjectsToCopy\_Source ( Name, ObjectType ) values ('kplus.dbo.ExternalConstantsMvtsT', 'Table') , ('kplus.dbo.ExternalConstantsT', 'Table') , ('kplus.dbo.ExternalDealsT', 'Table') , ('kplus.dbo.BasisSwapNRT', 'Table') , ('kplus.dbo.BalanceT', 'Table')

insert into DataMigration.SybtoSql\_ObjectsToCopy\_Source ( Name, ObjectType ) values ('master.dbo.syslogins', 'Logins' )

insert into DataMigration.SybtoSql\_ObjectsToCopy\_Source ( Name, ObjectType ) values ('kplus.dbo.sysusers', 'Users' )

The next call below sets the default mapping of source to destination. This means that for every source object, there is a destination object of the same name. CDMT supports source to destination table mapping allowing for tables from the source database to be mapped to different tables in the destination. See appendix for a more complete description.

exec DataMigration.MapDestinationFromSource

Once the call above is made and any additional mappings are applied, the queue is ready to be populated via the call below. The call to this stored procedure will populate the DataMigration.SybtoSql\_QueueDetail table. All rows will be deleted from the table by default. This behavior can be overridden by setting the @withTruncate parameter to 1 (0 by default).

exec DataMigration.PopulateQueue

Once the queue is populated, data can be copied. This is done by calling one or more of the procedures below. The parameter of 1 corresponds to the tnum column in the DataMigration.SybtoSql\_Config table.

exec DataMigration.SybtoSql\_CopyData 1

exec UserMigration.SybtoSql\_CopyLogins 1

exec UserMigration.SybtoSql\_CopyUsers 1

Progess can be monitored by selecting from the DataMigration.SybtoSql\_QueueDetail table during execution. See section below on Workflow states for an explanation of the table data.

select \* from DataMigration.SybtoSql\_QueueDetail

#### Workflow States

As the data copy progresses, each object in the queue moves through a series of states. The major states are represented below.

Waiting

BCP OUT

Drop Index

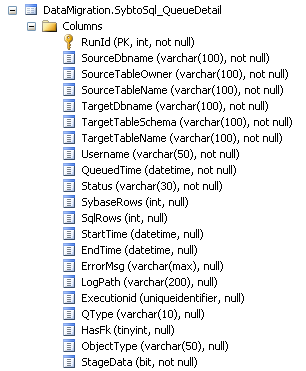
Bulk Insert

Add Index

Success Error

Each of entries in the DataMigration.SybtoSql\_QueueDetail table are keyed by the RunId column. Each item moves through the states above. Note, a transition to the error state could happen from any other state. See below for example data.

The full column list is below. Column names are fairly self-explanatory. The Status column will show the states above.



## Appendix A – Object Mapping Support

## Appendix B – Data Replacement Support

## Appendix C – Annoted DataMigration.SybtoSql\_CopyDataThread

This procedure is at the core of data migration. It orchestrates the data being copied out of Sybase and into SQL Server.

CREATE PROCEDURE [DataMigration].[SybtoSql\_CopyDataThread]

(

@tnum tinyint

, @datadir varchar(255)

, @bcppath varchar(255)

, @dbname sysname

, @tabowner sysname

, @tablename sysname

, @sybserver varchar(50)

, @sybid varchar(50)

, @interfacesfile varchar(255)

, @rowterminator varchar(255)

, @columnterminator varchar(255)

, @logdirectory varchar(255)

, @batchsize varchar(255)

, @numberofthreads int

, @defaultthreads int

, @targetdbname sysname

, @targettabowner sysname

, @targettablename sysname

)

as

set nocount on

declare @cmd nvarchar(600),

@cmdParamDef nvarchar(500),

@sybpass varchar(50),

@outrows int,

@inrows int,

@inrows\_after\_rep int,

@out int,

@runid int,

@pre\_cmd nvarchar(max),

@post\_before\_index\_cmd nvarchar(max),

@post\_after\_index\_cmd nvarchar(max),

@stage\_cmd nvarchar(max),

@stage\_data bit,

@sourceFilename varchar(255),

@targetFilename varchar(255),

@errormsg varchar(max)

set @cmdParamDef = N'@runId int, @processedRowCount bigint output'

BEGIN TRY

-- Check the if the status is ok to run

select @sybserver=syb\_server,@datadir = logpath

from DataMigration.SybtoSql\_Config where tnum=@tnum and qtype='BCP' and status='START'

if @@rowcount <> 1 return -200

--print 'Found configuration for tnum = ' + convert(varchar(40),@tnum)

-- Check if tables to be copied

exec DataMigration.SybtoSql\_GetTableFromQueue @qtype = 'BCP', @tnum = @tnum, @runid = @runid out

if @runid = -1 return -600

select @dbname = SourceDbname

, @tabowner = SourceTableOwner

, @tablename = SourceTableName

, @targetdbname = TargetDbname

, @targettabowner = TargetTableSchema

, @targettablename = TargetTableName

from DataMigration.SybtoSql\_QueueDetail

where RunId=@runid

--print 'Executing RunId ' + convert(varchar(50), @runid) + '...'

select @pre\_cmd='use ' + @dbname + ';' + pre\_cmd,

@post\_before\_index\_cmd = 'use ' + @dbname + ';' + post\_before\_index\_cmd,

@post\_after\_index\_cmd = 'use ' + @dbname + ';' + post\_after\_index\_cmd

from DataMigration.SybtoSql\_PrePost

where dbname=@dbname and tablename=@tablename and schemaname = @targettabowner

-- Init

select @sybid = SourceUser

from DataMigration.SybtoSql\_UserStore

where SourceDataServer = @sybserver

--and SourceDatabase = @dbname

if @sybid is null

begin

raiserror( 'Username not found.', 16, 1 )

return -800

end

exec DataMigration.GetCredentials @sybserver, @dbname, @sybid, @sourcePassword = @sybpass output

-- BCP out data from Sybase

set @cmd = @bcppath + ' ' + @dbname + '.' + @tabowner + '.' + @tablename +

' out "' + @datadir + @tablename +

'.bcpdata" -C -S' + @sybserver +

' -t"' + @columnterminator + '" -r"' + @rowterminator + '" ' +

' -U' + @sybid + ' -P' + @sybpass +

' -T 20480000 -c -b ' + @batchsize + ' > "' + @logdirectory + @tablename + '.log"'

print @cmd

update DataMigration.SybtoSql\_QueueDetail set Status = 'BCP Out',LogPath=@datadir where RunId=@runid

exec master..xp\_cmdshell @cmd ,NO\_OUTPUT

-- Verify Log file

set @cmd = '"' + @datadir + @tablename + '.log"'

exec DataMigration.VerifyBcpLog @cmd, @outrows = @outrows OUTPUT

update DataMigration.SybtoSql\_QueueDetail set SybaseRows = @outrows where RunId = @runid

-- Pre Command

if @pre\_cmd is not null

begin

update DataMigration.SybtoSql\_QueueDetail set Status = 'Pre Command' where RunId=@runid

exec sp\_executesql @pre\_cmd, @cmdParamDef, @runId = @runid, @processedRowCount = null

end

-- Save and Remove FK

--update DataMigration.SybtoSql\_QueueDetail set Status = 'Save And Remove FK' where RunId=@runid

--exec [DataMigration].[SaveAndRemoveFK] @dbname, @runid

-- Drop Indexex/PK/FK

update DataMigration.SybtoSql\_QueueDetail set Status = 'Drop Index' where RunId = @runid

exec DataMigration.GenerateScript @targettablename, @targetdbname, @targettabowner, 'Y', 'D', 'Y', 'N'

-- Truncate/Delete Table

set @cmd = 'select @out=count(\*)

from ' + @targetdbname + '.sys.foreign\_keys FK

inner join ' + @targetdbname + '.sys.tables T on T.object\_id=FK.referenced\_object\_id

and T.name=''' + @targettablename + '''

inner join sys.database\_principals p

on p.principal\_id=T.schema\_id and p.name=''' + @targettabowner + ''''

exec sp\_executesql @cmd,N'@out int OUTPUT',@out=@out OUTPUT

if @out = 0

set @cmd = 'truncate table ' + @targetdbname + '.[' + @targettabowner + '].[' + @targettablename + ']'

else

set @cmd = 'delete from ' + @targetdbname + '.[' + @targettabowner + '].[' + @targettablename + ']'

update DataMigration.SybtoSql\_QueueDetail set Status = 'Truncate/Delete' where RunId = @runid

exec sp\_executesql @cmd

if @@error <> 0

begin

RAISERROR ('Cmd failed - %' ,16,1,@cmd)

end

select @stage\_data = StageData

from DataMigration.SybtoSql\_QueueDetail

where RunId = @runid

if @stage\_data = 1

begin

-- Run Staging command

update DataMigration.SybtoSql\_QueueDetail set Status = 'Staging Data' where RunId = @runid

select @stage\_cmd = stage\_cmd

from DataMigration.SybtoSql\_PrePost

where dbname = @dbname

and tablename = @tablename

and schemaname = @targettabowner

exec sp\_executesql @stage\_cmd, @cmdParamDef, @runId = @runid, @processedRowCount = @inrows output

update DataMigration.SybtoSql\_QueueDetail set SqlRows = @inrows where RunId = @runid

end

else

begin

-- Bulk Insert

update DataMigration.SybtoSql\_QueueDetail set Status = 'Bulk Insert' where RunId = @runid

set @cmd = ' if exist "' + @datadir + @tablename + '.err\*"' + ' del "' + @datadir + @tablename + '.err\*";' +

' if exist "' + @datadir + @tablename + '.in\*"' + ' del "' + @datadir + @tablename + '.in\*";'

exec xp\_cmdshell @cmd,NO\_OUTPUT

set @cmd = 'use ' + @targetdbname + '; BULK INSERT [' + @targettabowner + '].[' + @targettablename +

'] from ''' + @datadir + @tablename + '.bcpdata''' +

' with (MAXERRORS=3000, codepage=''acp'',KEEPIDENTITY,KEEPNULLS, FIELDTERMINATOR = ''' + @columnterminator + ''',ROWTERMINATOR =''' + @rowterminator + ''', tablock ,batchsize = 500000, errorfile = ''' + @datadir + @tablename + '.err'')'

print @cmd

exec sp\_executesql @cmd

select @inrows = @@rowcount

update DataMigration.SybtoSql\_QueueDetail set SqlRows = @inrows where RunId = @runid

end

-- Post Before Command

if @post\_before\_index\_cmd is not null

begin

update DataMigration.SybtoSql\_QueueDetail set Status = 'Post Before Command' where RunId = @runid

exec(@post\_before\_index\_cmd)

end

-- Add Indexes/PK/FK

update DataMigration.SybtoSql\_QueueDetail set Status = 'Add Index' where RunId = @runid

exec DataMigration.GenerateScript @targettablename, @targetdbname, @targettabowner, 'Y', 'A', 'Y', 'N'

-- Add (Restore) Saved FK

--exec [DataMigration].[AddSavedFK] @dbname, @runid

-- Post After Command

if @post\_after\_index\_cmd is not null

begin

update DataMigration.SybtoSql\_QueueDetail set Status = 'Post After Command' where RunId = @runid

exec(@post\_after\_index\_cmd)

end

-- Verify if counts

if isnull(@inrows, -1) <> isnull(@outrows, -2)

begin

-- there are errors only if the replacement fail and the update is made just next line to "Replacing"

--update DataMigration.SybtoSql\_QueueDetail set Status = 'Error', ErrorMsg = 'Remote v/s Local rows do not match' where RunId = @runid

set @sourceFilename = @datadir + @tablename + '.err'

set @targetFilename = @datadir + @tablename + '.in'

-- if the ErrorMsg is not NULL it means there was an error or a warning

update DataMigration.SybtoSql\_QueueDetail set Status = 'Replacing', ErrorMsg = NULL where RunId = @runid

print 'Applying replacements to [' + isnull(@sourceFilename, 'Unknown') + '] -> [' + isnull(@targetFilename, 'Unknown') + '].'

exec DataMigration.ApplyReplacements @runid = @runid, @sourceFilename = @sourceFilename, @targetFilename = @targetFilename, @targetDbName = @targetdbname, @targetSchemaName = @targettabowner, @targetTableName = @targettablename, @columnTerminator = @columnterminator, @rowTerminator = @rowterminator, @rowsInserted = @inrows\_after\_rep output

if isnull((@inrows + isnull(@inrows\_after\_rep,0)), -1) <> isnull(@outrows, -2)

begin

select @errormsg = 'Remote v/s Local rows do not match. '

+ isnull(convert(varchar, (@inrows + @inrows\_after\_rep - @outrows)),'An unknown number of')

+ ' lines where not inserted.'

+ ' Un-inserted lines are placed into data file: "' + isnull(@sourceFilename,'Unknown') + '".'

if isnull(@inrows\_after\_rep,0) <> 0

select @errormsg = @errormsg + ' Note that ' + convert(varchar,@inrows\_after\_rep)

+ ' lines containing "Infinite" or "Not A Number field" values where replaced by 0.0 and inserted.'

else

print ' No replacement made.'

update DataMigration.SybtoSql\_QueueDetail set Status = 'Error', ErrorMsg = @errormsg where RunId = @runid

end

else

begin

set @cmd = 'if exist "' + @datadir + @tablename + '.bcpdata"' + ' del "' + @datadir + @tablename + '.bcpdata";' +

' if exist "' + @datadir + @tablename + '.log"' + ' del "' + @datadir + @tablename + '.log";' +

' if exist "' + @datadir + @tablename + '.in"' + ' del "' + @datadir + @tablename + '.in"'

exec xp\_cmdshell @cmd,NO\_OUTPUT

update DataMigration.SybtoSql\_QueueDetail set Status = 'Warning', ErrorMsg =

+ isnull(convert(varchar, @inrows\_after\_rep),'Unknown number')

+ ' lines containing "Infinite" or "Not A Number field" values where replaced by 0.0'

+ ' Original lines are placed into data file: "' + isnull(@sourceFilename,'Unknown') + '"'

end

end

else

begin

set @cmd = 'if exist "' + @datadir + @tablename + '.bcpdata"' + ' del "' + @datadir + @tablename + '.bcpdata";' +

' if exist "' + @datadir + @tablename + '.log"' + ' del "' + @datadir + @tablename + '.log";' +

' if exist "' + @datadir + @tablename + '.err"' + ' del "' + @datadir + @tablename + '.err"'

exec xp\_cmdshell @cmd,NO\_OUTPUT

update DataMigration.SybtoSql\_QueueDetail set EndTime = getdate(), Status='Success' where RunId = @runid

end

END TRY

BEGIN CATCH

select @errormsg =

'ERROR\_NUMBER: ' + isnull(convert(varchar(10),ERROR\_NUMBER()),-1) +

' ERROR\_SEVERITY: ' + isnull(convert(varchar(10),ERROR\_SEVERITY()),-1) +

' ERROR\_STATE: ' + isnull(convert(varchar(10),ERROR\_STATE()),-1) +

' ERROR\_LINE: ' + isnull(convert(varchar(10),ERROR\_LINE()),-1) +

' ERROR\_PROCEDURE: ' + isnull(ERROR\_PROCEDURE(),'') +

' ERROR\_MESSAGE: ' + isnull(ERROR\_MESSAGE(),'')

update DataMigration.SybtoSql\_QueueDetail set Status = 'Error', ErrorMsg = @errormsg where RunId = @runid

END CATCH;

GO

## Appendix D – Object Listing

