Example Instructional Files: Code Artifacts
from
SAP® Data Services: The Comprehensive Guide

This supplement contains three documents that each provide a set of code artifacts to enable examples that were introduced in the book to work in your own environment. Each document contains a readme instructional file to guide you through the example's purpose, configuration, and execution steps. Please note that each example is provided for demonstration purpose as-is and should go through testing prior to execution within a production environment.
1 Implementing Source-Based CDC in SAP Data Services

Change data capture (CDC) is a process that involves identifying and handling only those records that have changed from a source table to a target table. The applications are wide ranging, but the benefits of CDC include significant savings in the time and cost of processing large amounts of data from a source. There are two types of CDC: target-based and source-based. We will discuss the latter in this example.

This document will provide insight as to how source-based CDC is implemented in SAP Data Services, including configuring the source database system (i.e., Microsoft SQL Server) to use CDC. We assume that fresh installations of SAP Data Services 4.2 and Microsoft SQL Server 2012 Enterprise have been completed, and that everything has been configured properly.
Implementing Source-Based CDC in SAP Data Services

Execute the following Transact-SQL commands located in cdc_create_enable_database.sql to create your database, then enable CDC on it.

This stored procedure enables CDC on the active database, but additional steps are required for full functionality.

In this example, we use an EMPLOYEE table. Execute create_employee_table.sql to bring this into the CDC_RESOURCE database.

It is important to not load your data into the table before enabling CDC on it.

This T-SQL (located in cdc_enable_table.sql) will execute a stored procedure on the table to enable CDC for it.

@role_name allows you to specify parameters for who can access the CDC system table.

We see that SQL Server jobs have started that perform CDC operations on this table.
After enabling CDC, a new system table `dbo_EMPLOYEE_CT` (owned by `cdcrather than dbo) exists, which will store any changes made to `dbo.EMPLOYEE`.

`dbo.EMPLOYEE` only shows the most recent version of any record that’s been changed.

A select statement showing another way to verify whether or not CDC has been successfully enabled on your table.

The ‘1’ indicates that the `EMPLOYEE` table is CDC-enabled.
Only after enabling CDC on the table can we insert records into it. This example populates the employee table with randomly generated data.

CDC_DEMO.atl has been included, showing the objects used in the following section. (The password for the file is “Method360”.)

Click the Datastores tab in the Central Object Repository.

Click New to add a data source.
The No CDC dropdown must be changed to say Native CDC or Replication Server CDC if a replication server is in use.

Configure additional options as necessary.

Turning on CDC when importing a DataStore only shows CDC-enabled tables afterwards. Here, a DataStore has been imported twice (once with CDC, once without) to enable access to both CDC and non-CDC tables.

Both DataStores as they appear in the Local Object Library. Drag the table into your project workspace.
Implementing Source-Based CDC in SAP Data Services

Double-click the source CDC object in the workspace.

Check these configuration options. This will delete records from the CDC system table after they have been loaded into the target table to ensure that it is not possible to attempt to load them again during a subsequent job execution.

The DI_OPERATION_TYPE column indicates what type of operation was performed here against the row. Indicating Insert, Update, or Delete with I, U, or D.

The CDC-enabled table in our project workspace as a source object.

A sample data flow showing a Map_CDC_Operation transform.

The transform handles how CDC-tracked data from the source is inserted into the target.
Implementing Source-Based CDC in SAP Data Services

The employee data is filtered to check for changes.

The CDC columns are automatically recognized by the MAP_CDC_OPERATION transform.

The first time the job is executed, all rows are passed into the target table, and a checkpoint is made.

If we insert 50 more rows into the source table and run this job again, only those newly inserted rows will be passed to the target table due to the checkpoint we set up in a previous step, and the behavior specified by the CDC_MAP_OPERATION transform.
2 Configure a Job in the Management Console

When configuring an SAP Data Services job in the management console, SAP provides an “Export Execution Command” feature, which enables automated job execution from a CLI rather than through the Data Services Management Console/Designer tool. This feature is useful for the purposes of job integration into an enterprise scheduler such as Redwood/TWS/Maestro/UC4. However, the CLI job execution tool fails to print relevant log information out to the screen or return a useful exit status for the scheduler to utilize in the event of a job execution failure.

This shell script provides the aforementioned missing functionality so that the enterprise scheduler can capture log file data and error codes in order for them to be passed on to relevant support personnel. There are two primary versions of this document: one to operate on a Windows environment and one for Unix/Linux. Both provide configuration instructions and show how a return code could be passed back into an enterprise scheduler.

**Note**

Job design still requires proper raising of exceptions which may require use of the RAISE_EXCEPTION_EXT function so that the exception is properly returned after the AL_RWJOBBLAUNCHER execution is finished.

### 2.1 Windows Version

```bash
@ECHO OFF
::INSERT THE CONTENTS OF YOUR DATA SERVICES GENERATED .BAT FILE BETWEEN THE FOLLOWING LINES

:: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

:: DO NOT MODIFY THE FOLLOWING COMMAND
set JOBERROR=%ERRORLEVEL%

:: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```
:: SET THE REST OF YOUR VALUES HERE
:: SET PATH TO YOUR LOGGING DIRECTORY.
:: EX:
:: set DSJOB_LOGDIRECTORY="C:\ProgramData\SAP BusinessObjects\Data Services\log\jobserver"

set DSJOB_LOGDIRECTORY=C:\YOUR\PATH\HERE

:: SET UNIQUE IDENTIFIER GIVEN IN THE -G FLAG OF YOUR DATA SERVICES GENERATED .TXT FILE
:: EX IF: -G"81bfdfce_f49e_4577_8b6e_b09449297087"
:: THEN: set DSJOB_UNIQUEID=81bfdfce_f49e_4577_8b6e_b09449297087

set DSJOB_UNIQUEID=INSERT_GUID_HERE

::OPTIONAL: MODIFY SCREEN OUTPUT FORMATTING BELOW AS DESIRED

:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cd %DSJOB_LOGDIRECTORY%
for /f "tokens=*" %%G in ('dir trace*%DSJOB_UNIQUEID%*.txt /b /a-d /od') do SET NWST_TRACE=%%G
for /f "tokens=*" %%G in ('dir error*%DSJOB_UNIQUEID%*.txt /b /a-d /od') do SET NWST_ERROR=%%G
@if %JOBERROR% EQU 0 (  
   echo ------------------------------
   echo TRACE LOG:
   echo ------------------------------
   echo type %NWST_TRACE%
   echo ------------------------------
   echo JOB SUCCESSFUL
   echo ------------------------------
) ELSE (  
   echo ------------------------------
   echo TRACE LOG:
   echo ------------------------------
   echo type %NWST_TRACE%
   echo ------------------------------
   echo ERROR LOG:
   echo ------------------------------
   echo type %NWST_ERROR%
   echo ------------------------------
   echo JOB FAILED: ERROR CODE %JOBERROR%

:: SET PATH TO YOUR LOGGING DIRECTORY.
:: EX:
:: set DSJOB_LOGDIRECTORY="C:\ProgramData\SAP BusinessObjects\Data Services\log\jobserver"

set DSJOB_LOGDIRECTORY=C:\YOUR\PATH\HERE

:: SET UNIQUE IDENTIFIER GIVEN IN THE -G FLAG OF YOUR DATA SERVICES GENERATED .TXT FILE
:: EX IF: -G"81bfdfce_f49e_4577_8b6e_b09449297087"
:: THEN: set DSJOB_UNIQUEID=81bfdfce_f49e_4577_8b6e_b09449297087

set DSJOB_UNIQUEID=INSERT_GUID_HERE

::OPTIONAL: MODIFY SCREEN OUTPUT FORMATTING BELOW AS DESIRED

:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cd %DSJOB_LOGDIRECTORY%
for /f "tokens=*" %%G in ('dir trace*%DSJOB_UNIQUEID%*.txt /b /a-d /od') do SET NWST_TRACE=%%G
for /f "tokens=*" %%G in ('dir error*%DSJOB_UNIQUEID%*.txt /b /a-d /od') do SET NWST_ERROR=%%G
@if %JOBERROR% EQU 0 (  
   echo ------------------------------
   echo TRACE LOG:
   echo ------------------------------
   echo type %NWST_TRACE%
   echo ------------------------------
   echo JOB SUCCESSFUL
   echo ------------------------------
) ELSE (  
   echo ------------------------------
   echo TRACE LOG:
   echo ------------------------------
   echo type %NWST_TRACE%
   echo ------------------------------
   echo ERROR LOG:
   echo ------------------------------
   echo type %NWST_ERROR%
   echo ------------------------------
   echo JOB FAILED: ERROR CODE %JOBERROR%

:: SET THE REST OF YOUR VALUES HERE
:: SET PATH TO YOUR LOGGING DIRECTORY.
:: EX:
:: set DSJOB_LOGDIRECTORY="C:\ProgramData\SAP BusinessObjects\Data Services\log\jobserver"

set DSJOB_LOGDIRECTORY=C:\YOUR\PATH\HERE

:: SET UNIQUE IDENTIFIER GIVEN IN THE -G FLAG OF YOUR DATA SERVICES GENERATED .TXT FILE
:: EX IF: -G"81bfdfce_f49e_4577_8b6e_b09449297087"
:: THEN: set DSJOB_UNIQUEID=81bfdfce_f49e_4577_8b6e_b09449297087

set DSJOB_UNIQUEID=INSERT_GUID_HERE

::OPTIONAL: MODIFY SCREEN OUTPUT FORMATTING BELOW AS DESIRED

:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cd %DSJOB_LOGDIRECTORY%
for /f "tokens=*" %%G in ('dir trace*%DSJOB_UNIQUEID%*.txt /b /a-d /od') do SET NWST_TRACE=%%G
for /f "tokens=*" %%G in ('dir error*%DSJOB_UNIQUEID%*.txt /b /a-d /od') do SET NWST_ERROR=%%G
@if %JOBERROR% EQU 0 (  
   echo ------------------------------
   echo TRACE LOG:
   echo ------------------------------
   echo type %NWST_TRACE%
   echo ------------------------------
   echo JOB SUCCESSFUL
   echo ------------------------------
) ELSE (  
   echo ------------------------------
   echo TRACE LOG:
   echo ------------------------------
   echo type %NWST_TRACE%
   echo ------------------------------
   echo ERROR LOG:
   echo ------------------------------
   echo type %NWST_ERROR%
   echo ------------------------------
   echo JOB FAILED: ERROR CODE %JOBERROR%
```bash
#!/bin/bash

#INSERT THE CONTENTS OF YOUR DATA SERVICES GENERATED .sh FILE BETWEEN
# THE FOLLOWING LINES
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# DO NOT MODIFY THE FOLLOWING COMMAND
export ERRORLEVEL="$?"
#SET THE REST OF YOUR VALUES HERE
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# SET PATH TO YOUR LOGGING DIRECTORY.
# EX: export DSJOB_LOGDIRECTORY="/build/dshana/dataservices/log/
# Server1/hanarepo__dataservices"
export DSJOB_LOGDIRECTORY="YOUR/PATH/HERE"

# SET UNIQUE IDENTIFIER GIVEN IN THE -G FLAG OF YOUR DATA SERVICES
# GENERATED .sh FILE
# EX: -G"81bfdfce_f49e_4577_8b6e_b09449297087"
# THEN: export DSJOB_UNIQUEID="81bfdfce_f49e_4577_8b6e_
b09449297087"
export DSJOB_UNIQUEID="UNIQUE_ID_HERE"

#OPTIONAL: MODIFY SCREEN OUTPUT FORMATTING BELOW AS DESIRED
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

cd $DSJOB_LOGDIRECTORY
NST_TRACE=$(ls -t *trace*$DSJOB_UNIQUEID* | head -1);
NST_ERR=$(ls -t *error*$DSJOB_UNIQUEID* | head -1);
if [ "$ERRORLEVEL" = 0 ];then
  echo "-----------------------------"
echo "TRACE LOG:"
echo "-----------------------------"
```
cat $NST_TRACE
echo ''
else
  echo ''
  cat $NST_TRACE
  echo ''
  echo "$TRACE LOG:"
  echo ''
  cat $NST_TRACE
  echo ''
  echo "$ERROR LOG:"
  echo ''
  cat $NST_ERR
  echo ''
  echo "$JOB FAILED: ERROR CODE $ERRORLEVEL"
  echo ''
  fi

exit $ERRORLEVEL

2.3 Configuration by Section (Windows)

First, we’ll go over the steps you need for configuration in Windows.

2.3.1 Header

Copy the contents of your old .BAT/.sh file provided by the SAP Data Services “Export Execution Command” function into the empty space provided in the script header.

The Windows environment references a .txt file with additional execution parameters. Ensure that this file will be accessible from our new script (i.e., both files are in the same folder).
We will reference this text file in the next step for the logging directory and log GUID.

@ECHO OFF
:: INSERT THE CONTENTS OF YOUR DATA SERVICES GENERATED .BAT FILE BETWEEN
:: THE FOLLOWING LINES
:: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
:: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

2.3.2 Setting Log Directory and GUID Pointers

1. Set the path to your logging directory.

   **Note**
   
   As mentioned in the previous step, the .BAT file generated by Data Services only gives
   you the logging path to the job server folder.

2. Navigate to this folder and find the subdirectory that corresponds with the
   repository this job uses.

3. Copy and paste this full path into the DSJOB_LOGDIRECTORY variable.

4. Locate the name of the GUID given with flag –G in the .BAT file generated by
   Data Services.

5. Copy and paste this GUID into the DSJOB_UNIQUEID variable.

   Configuration examples for this section are included in the comments above the
   dummy values for each variable. The comments provide good examples as to how
   the variables are configured per platform.

   :: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   :: SET THE REST OF YOUR VALUES HERE
   :: SET PATH TO YOUR LOGGING DIRECTORY.
   :: EX:
   :: set DSJOB_LOGDIRECTORY="C:\ProgramData\SAP BusinessObjects\Data
   Services\log\jobserver"
   :: set DSJOB_LOGDIRECTORY=C:\YOUR\PATH\HERE
   :: SET UNIQUE IDENTIFIER GIVEN IN THE -G FLAG OF YOUR DATA SERVICES
   GENERATED .TXT FILE
   :: EX IF: -G"81bfdfe_f49e_4577_8b6e_b0944927087"
   :: THEN: set DSJOB_UNIQUEID=81bfdfe_f49e_4577_8b6e_
   b0944927087
   :: set DSJOB_UNIQUEID=INSERT_GUID_HERE
2.4  Configuration by Section (Unix/Linux)

Now we’ll go over the steps you need for configuration in Unix/Linux.

2.4.1  Header
The Unix environment includes all execution parameters, including GUID in the .sh file Data Services generated, and doesn’t reference a text file.

The log directory is specified up to the Job Server level in one of the flags in the .BAT/.sh script.

Navigate to this folder and find the subdirectory that corresponds with the repository this job uses.

Copy the path to this repository log folder for use in the next step.

#INSERT THE CONTENTS OF YOUR DATA SERVICES GENERATED .sh FILE BETWEEN THE FOLLOWING LINES
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

2.4.2  Setting Log Directory and GUID Pointers

1. Set the path to your logging directory.

Note
As mentioned in the previous step, the .sh file generated by Data Services only gives you the logging path to the job server folder.

2. Navigate to this folder and find the subdirectory that corresponds with the repository this job uses.

3. Copy and paste this full path into the DSJOB_LOGDIRECTORY variable.

4. Locate the name of the GUID given with flag –G in the .sh file generated by Data Services.

5. Copy and paste this GUID into the DSJOB_UNIQUEID variable.

Configuration examples for this section are included in the comments above the dummy values for each variable. The comments provide good examples as to how the variables are configured per platform.
Here is an example of successful job execution:

> win_dsjob.bat

---

**TRACE LOG:**

---

ECHO is off.

(14.2) 07-15-14 18:12:25 (6884:7808) JOB: Reading job <d1bf35b9_f567_4cff_acf8_5517c9c63738> from the repository; Server version is <14.2.2.446>; Repository version is <14.2.2.0000>.

(14.2) 07-15-14 18:12:25 (6884:7808) JOB: Current directory of job <d1bf35b9_f567_4cff_acf8_5517c9c63738> is <C:\Program Files (x86)\SAP BusinessObjects\Data Services\bin>.

(14.2) 07-15-14 18:12:26 (6884:7808) JOB: Starting job on job server host <WIN-7V7IQVRM0TN>, port <3010>.

(14.2) 07-15-14 18:12:27 (6884:7808) JOB: Job <LOAD_DW> of runid <2014071518122668847808> is initiated by user <SYSTEM>.


(14.2) 07-15-14 18:12:27 (6884:7808) JOB: Job <LOAD_DW> is started.

(14.2) 07-15-14 18:12:29 (2024:6860) DATAFLOW: Process to execute data flow <Load_DW> is started.

(14.2) 07-15-14 18:12:29 (2024:6860) DATAFLOW: Data flow <Load_DW> is
Configure a Job in the Management Console

started.
(14.2) 07-15-14 18:12:29 (2024:6860) DATAFLOW: Cache statistics determined that data flow <Load_DW> uses 0 caches with a total size of 0 bytes, which is less than (or equal to) 3757047808 bytes available for caches in virtual memory. Data flow will use IN MEMORY cache type.
(14.2) 07-15-14 18:12:39 (2024:6860) DATAFLOW: Data flow <Load_DW> is terminated due to error <70401>.
(14.2) 07-15-14 18:12:40 (6884:7808) JOB: Job <LOAD_DW> has completed successfully.
ECHO is off.
-----------------------------------------------
JOB SUCCESSFUL
-----------------------------------------------

Here is an example of failed job execution:

> win_dsjob.bat

-----------------------------------------------
TRACE LOG:
-----------------------------------------------
ECHO is off.
(14.2) 07-15-14 18:12:25 (6884:7808) JOB: Reading job <d1bf35b9_f567_4cfe_acf8_5517c963738> from the repository; Server version is <14.2.2.446>; Repository version is <14.2.2.0000>.
(14.2) 07-15-14 18:12:25 (6884:7808) JOB: Current directory of job <d1bf35b9_f567_4cfe_acf8_5517c963738> is <C:\Program Files (x86)\SAP BusinessObjects\Data Services\bin>.
(14.2) 07-15-14 18:12:27 (6884:7808) JOB: Job <LOAD_DW> of runid <2014071518122668847808> is initiated by user <SYSTEM>.
(14.2) 07-15-14 18:12:27 (6884:7808) JOB: Job <LOAD_DW> is started.
(14.2) 07-15-14 18:12:29 (2024:6860) DATAFLOW: Process to execute data flow <Load_DW> is started.
(14.2) 07-15-14 18:12:29 (2024:6860) DATAFLOW: Data flow <Load_DW> is started.
determined that data flow <Load_DW> uses 0 caches with a total size of 0 bytes, which is less than (or equal to) 3757047808 bytes available for caches in virtual memory. Data flow will use IN MEMORY cache type.

(14.2) 07-15-14 18:12:39 (2024:6860) DATAFLOW: Data flow <Load_DW> is terminated due to error <70401>.
(14.2) 07-15-14 18:12:40 (6884:7808) JOB: Job <LOAD_DW> is terminated due to error <70401>.
ECHO is off.

-------------------------------
ERROR LOG:
-------------------------------
ECHO is off.
(14.2.4) 02-18-15 10:05:34 (E) (4900:7644) SYS-170101: The system encountered an exception and cannot process this action. This message contains some internal system details which have been hidden for security. If you need to see the full contents of the original message, ask your administrator to assign additional privileges to your account.
ECHO is off.

-------------------------------
JOB FAILED: ERROR CODE 923
-------------------------------

%ERRORLEVEL% is a system environment variable. It can be interrogated by the enterprise scheduling software as well as shown from the command line as follows:

>echo %ERRORLEVEL%
923

923 represents the execution return code that would be evaluated by the enterprise scheduling software to determine success or failure of the job/job step.
3 Import and Use the SOAP Data Flow

A frequent data provisioning task includes having to interact with an application interface to extract or load data. Many times, this application interface comes in the form of web services. SAP Data Services offers several methods to interact with web services. This overview will guide you on importing a SOAP demonstration data flow into Data Services Designer. Please see Chapter 5, Section 5.6 for more background information.

1. Go to TOOLS • IMPORT from File to import the ATL.
2. Navigate to the location of the ATL and enter the password.
This demo uses a SOAP DataStore; the DataStore contains the available methods for that web service. It calls the web service through a function that’s embedded in a query transform, outputting to a nested schema. The second query transform un-nests the data so it can be loaded into a template table.
This demo uses a SOAP DataStore; the DataStore contains the available methods for that web service. It calls the web service through a function that’s embedded in a query transform and uses inputs (zip code) from the nested input schema, outputting to a nested schema. The second query transform un-nests the data so it can be loaded into a template table.