Proven Practice

How to define and use UdBR for IBM Cognos Controller

Product(s): IBM Cognos Controller

Area of Interest: Financial Management

Copyright and Trademarks

Licensed Materials - Property of IBM.

© Copyright IBM Corp. 2012

IBM, the IBM logo, and Cognos are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be

trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

While every attempt has been made to ensure that the information in this document is accurate and complete, some typographical errors or technical inaccuracies may exist. IBM does not accept responsibility for any kind of loss resulting from the use of information contained in this document. The information contained in this document is subject to change without notice.

This document is maintained by the **IBM Business Analytics Proven Practices** team. You can send comments, suggestions, and additions to cscogpp@ca.ibm.com.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license there from.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Table of Contents

INTRODUCTION 7

PURPOSE OF DOCUMENT 7

APPLICABILITY 7

ASSUMPTIONS 7

UDBR STEP BY STEP 7

CREATE AN AJT AND DEFINE A CONTROL TABLE 7

CREATE THE BUSINESS RULE 8

VALIDATE THE BUSINESS RULE 11

CREATE THE COMPANY SUBSET 12

DEFINE A UDBR JOB 13

MAPPING OF THE UDBR JOB 14

RUNNING OF THE BUSINESS RULES 15

PYTHON CODE 15

NARRATIVE 15

PRINT 18

VARIABLES 18

STRING COMMANDS – CONCATENATE 18

STRING COMMANDS – CONVERT TO STRING 18

STRING COMMANDS – CONVERT TO INT 19

LOGICAL STATEMENTS – IF COMMAND 19

LOGICAL STATEMENTS – IF ELSE COMMAND 20

LOGICAL STATEMENTS – WHILE STATEMENTS 20

LOGICAL STATEMENTS – FOR STATEMENTS 21

DEFINING FUNCTIONS 21

UDBR CREATION AND PROCESS 22

UDBR CREATION 22

THE SOURCE & LOAD 22

THE DEFAULT VALUES : 23

THE TARGET & SAVE: 23

THE CALCULATION: 24

THE PRE-DEFINED OBJECTS: 24

UDBR PROCESS 25

UDBR FUNCTIONS 31

RUN TIME FUNCTIONS 31

SOURCE FUNCTIONS 35

SOURCE FUNCTIONS – SET FUNCTIONS 35

SOURCE FUNCTIONS – GROUP BY FUNCTIONS 40

SOURCE FUNCTIONS – LOAD DATA: 42

SOURCE FUNCTIONS EXAMPLE: 44

TARGET FUNCTIONS 45

TARGET FUNCTIONS – SET FUNCTIONS 45

TARGET FUNCTIONS - CALCULATIONS: 51

TARGET FUNCTIONS – SAVE TO CONTROLLER: 52

TARGET FUNCTIONS EXAMPLE :..... 53

FILTER OBJECTS: 54

WRITER FUNCTIONS: 55

UDBR LIMITATIONS..... 56

UDBR VALIDATION 56

UDBR EXAMPLES..... 57

SPLIT AN AMOUNT BY USING A FIX PERCENTAGE. 57

CALCULATE INTERCOMPANY PROFIT MARGIN PER COUNTERPART 58

CALCULATE THE DEPRECIATION OF AN ASSET..... 61

KEEP ONLY POSITIVE OR NEGATIVE AMOUNTS..... 63

YTD AVERAGE BASED ON NUMBER OF SPENT DAYS..... 65

QTD AVERAGE BASED ON NUMBER OF SPENT DAYS 65

HANDLING OF STRUCTURAL CHANGES DURING THE YEAR..... 65

Introduction

Purpose of Document

A new feature for IBM Cognos Controller 10.1 is the use of User defined Business Rules (UdBR). The new feature allows the following:

Possibility to define your own customized calculations, fully integrated in the consolidation process with flexibility on when they are executed. Way of "automating" manual journals.

Enable customers and projects to solve specific needs outside release cycles.

Example scenarios:

- Calculate and store the impact on net income and minorities of structural changes during the year
- Calculate Intercompany Profit margins per counterpart,….
- A complement, but not a change, to Controller's application approach (rich feature set in-the-box).
- Must be used only for calculations that can not be done by using the standard calculations engine (currency conversion, intercompany eliminations, acquisition calculations, allocations, AFC)

Applicability

From IBM Cognos Controller 10.1

Assumptions

User defined Business rules (UdBR) required the user is familiar with account and company structures in IBM Cognos Controller. The UdBR are calculations that pull information from the controller database through account, company, period, actuality, etc. dimensions.

UdBR step by step

Create an AJT and define a Control table

This step must be done only if you decide to book the result of the UdBR in an Automatic journal.If you don't do it the values that are booked in an automatic journal by UdBR, won't be consolidated.

In Cognos Controller click **Maintain > Configuration > Define > Contribution versions - Automatic journal types.**

- Create an automatic journal other than number 1-99 For example: BR
- Don't forget to add it in relevant contribution versions.
- Define the automatic journal type in Maintain > Configuration > Automatic Journals > Define.
- Give a code (can not start with E) and a name. Enter in the field Automatic Journal Type the automatic journal you have created previously. For example: BR
- Select category 1 and activate it
- In **Maintain > Configuration > Automatic Journals > Control tables**, don't specify any accounts and any journal number in the control table (must be set in the business rule).

Create the Business rule

Create the User-defined business rules in the menu **Maintain** > **Business Rules**

The business rule (script) uses the python language which is a powerful programming language easy to learn. To build your own rule more easily you can get some basic knowledge see **http://docs.python.org/tutorial/**

The script is based on functions. See "UDBR creation and functions" section.

The script must include a source definition and a target definition as a minimum. The source defines the data that should be run in the consolidation. The target defines the data that is created and stored in the consolidation calculations. There are also filter definitions to use in the source and the target definitions, as well as pre-defined model functionality.

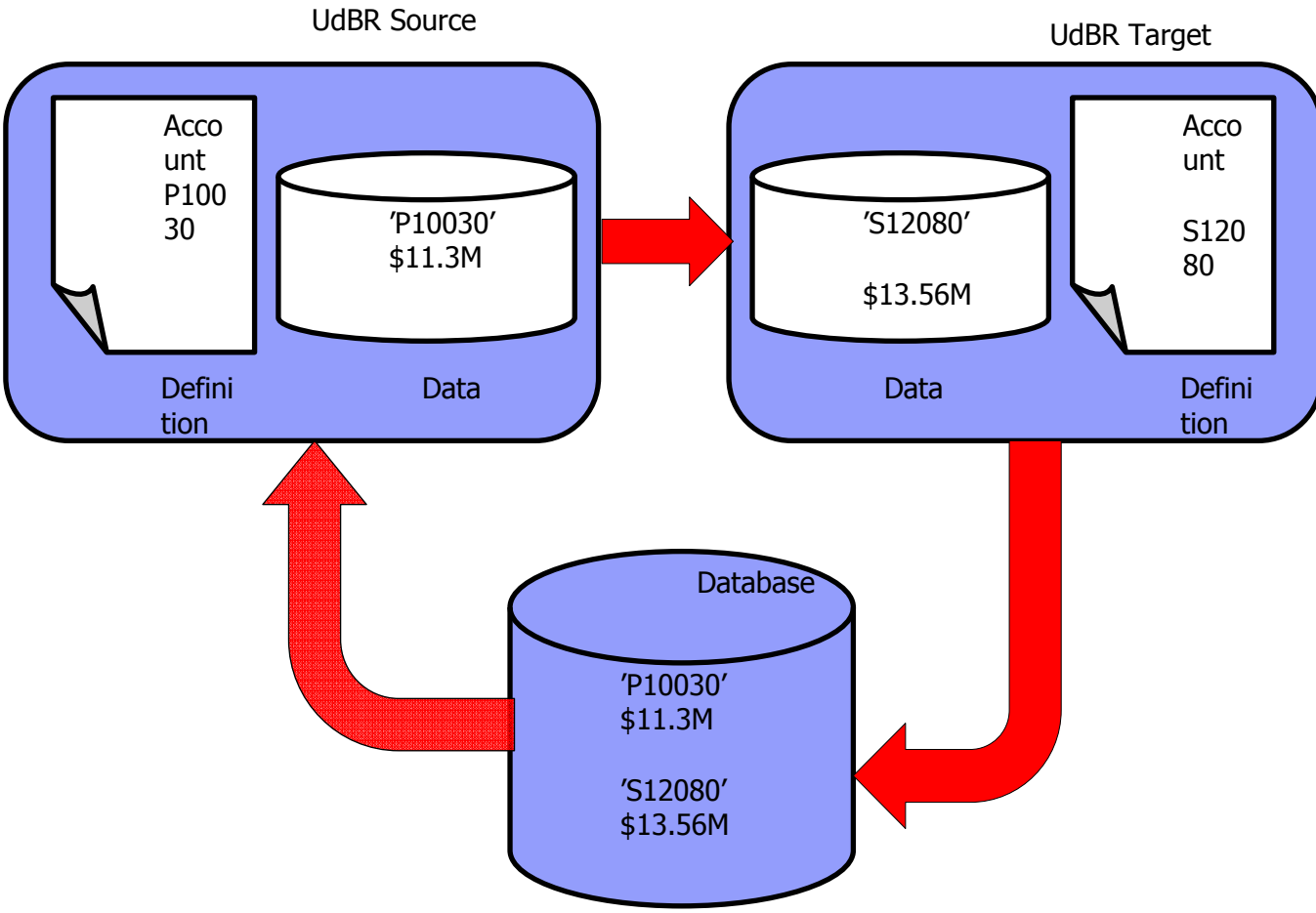


Figure 1 shows the flow UdBR from Source to the target within the IBM Cognos Controller database

A Business Rule consists of "code lines" (python language), one per row. A range of "financial style" functions have been created to retrieve period data, %, to make calculations, to store data …

Several rules can be created, for different purposes.

Name - English:

Move Data

Script:

```
#The purpose of this script is to copy the value from one account (SourceAccount)
#and move it to another account (TargetAccount)
#On copy the script multiplies the value by the multiplier
#This script was created by James Melville
#Created 2nd July 2012

#Define variables used in the script.
#Define the accounts we want to move data from and to.
SourceAccount = 'P10030'
TargetAccount = 'S12080'

#Define the multiplier we will multiply the value by.
Multiplier = 1.2

#Main part of the script
#Create the source definition of the data required
source1 = Creator.createSource()
source1.setAccount(SourceAccount)

#Load the data into the rule
source1.loadData()

#Now create the Target Values
target1= Creator.createTarget()
target1.setAccount(TargetAccount)
target1.addDataWithFactor(source1, Multiplier)

#Save the values back to Controller
target1.mergeAndSaveData()
```

Figure 2 shows sample UdBr code using python programming language

Validate the Business rule

You can make a validation of the UdBR by clicking the "validate all" button at the bottom of the UdBR define window.

You can make a preview of the result by running the script (note that the result will be stored!).After running it, you will get a log file with all the information you requested to be displayed with the function "print" in the script.

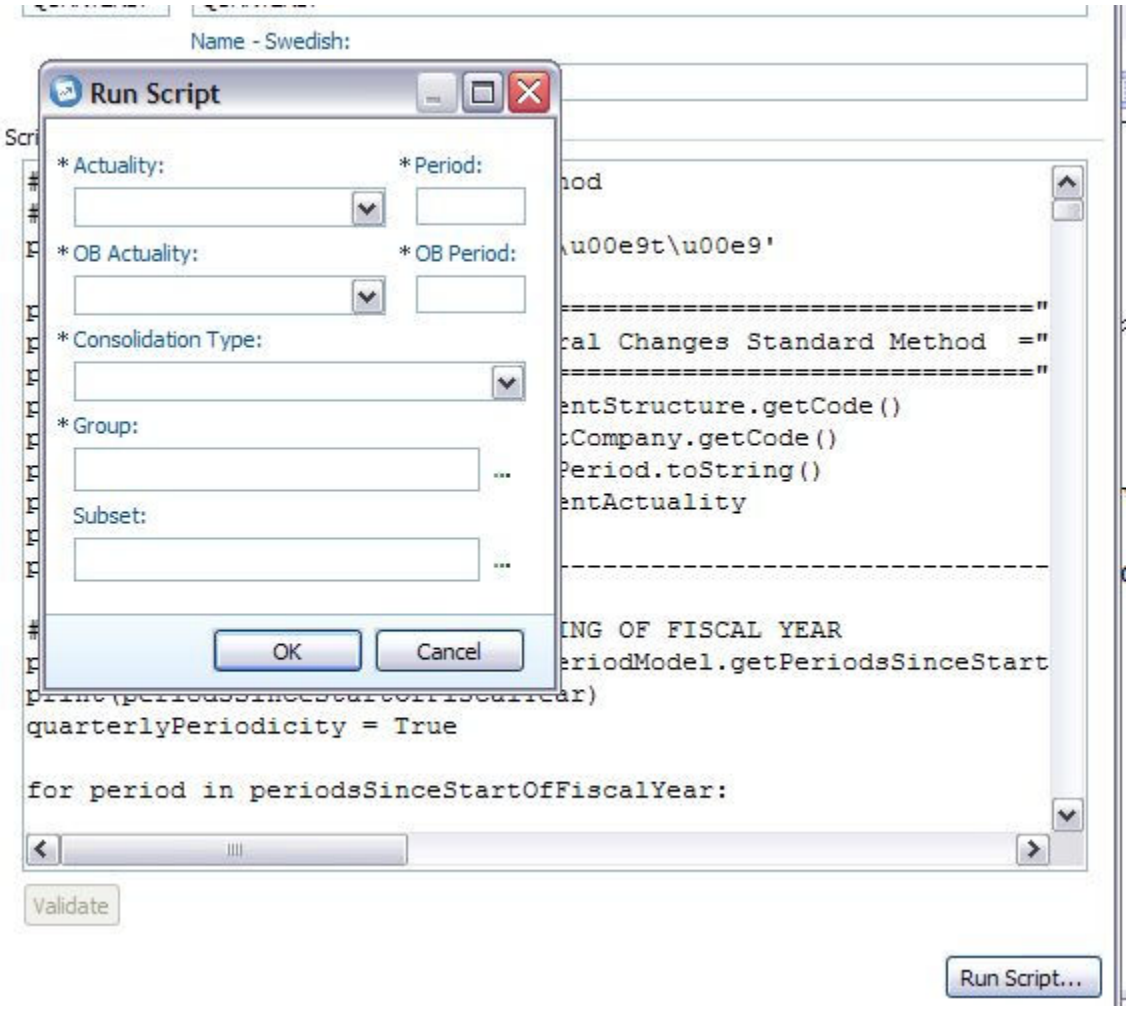


Figure 3 shows the running of the UdBR Script

Create the company subset

In **maintain > subset define,** define the companies that you would the rule will run against.

- Select "add a new row"
- Choose static selection or all entities
- If you select all entities, all companies and groups will be selected and will appear in the definition tab
- If you select static selection, you must add the companies and/or groups by editing the list of objects in the definition tab

Limitation: Don't select multi owned companies as UdBR don't handle it

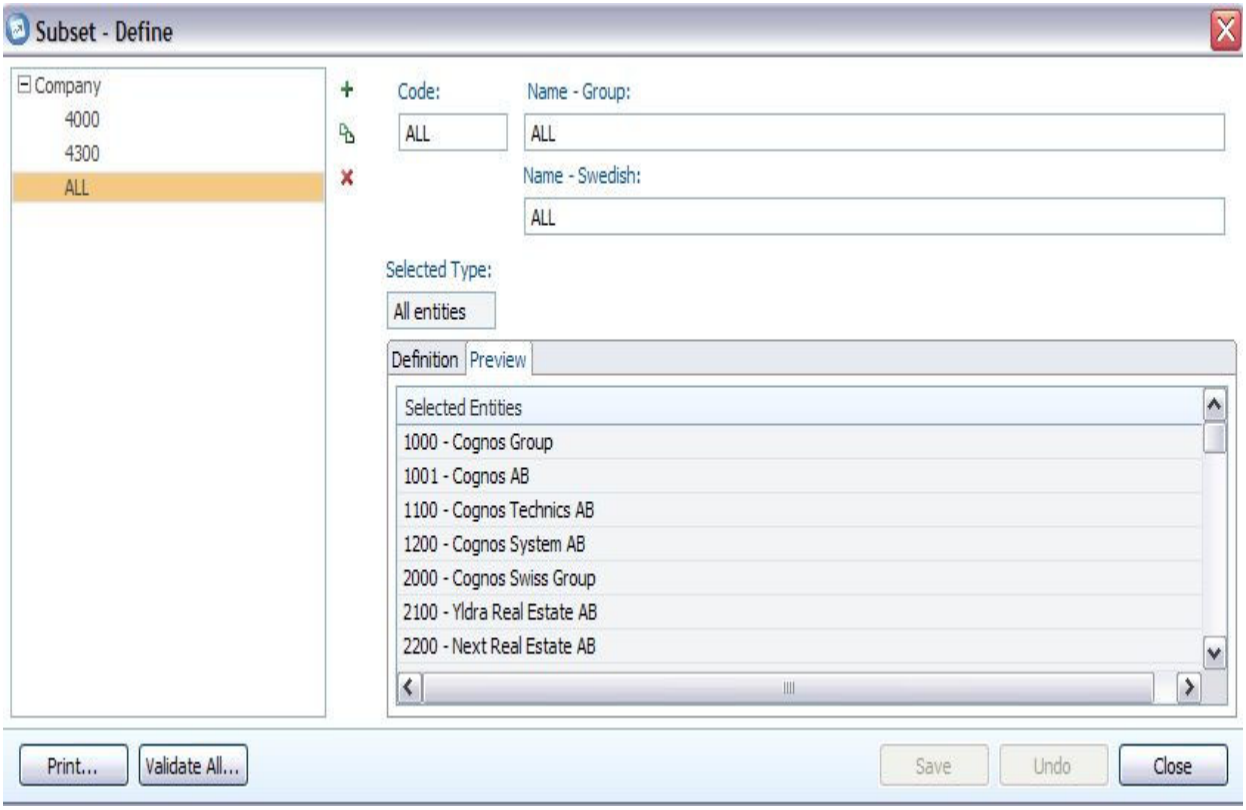


Figure 4 shows subset for UDBR parameters

Define a UdBR job

In **maintain >jobs>define** create a batch of rules and the sequence in which they should run. Associate each of these rules with a company subset.

- Go in maintain >jobs>define
- Select "add a new row" to create a job
- Select the business rules you want to set in the job and mention for each of them for which company subset you want to run it
- The business rules will be run in the order specified in the job

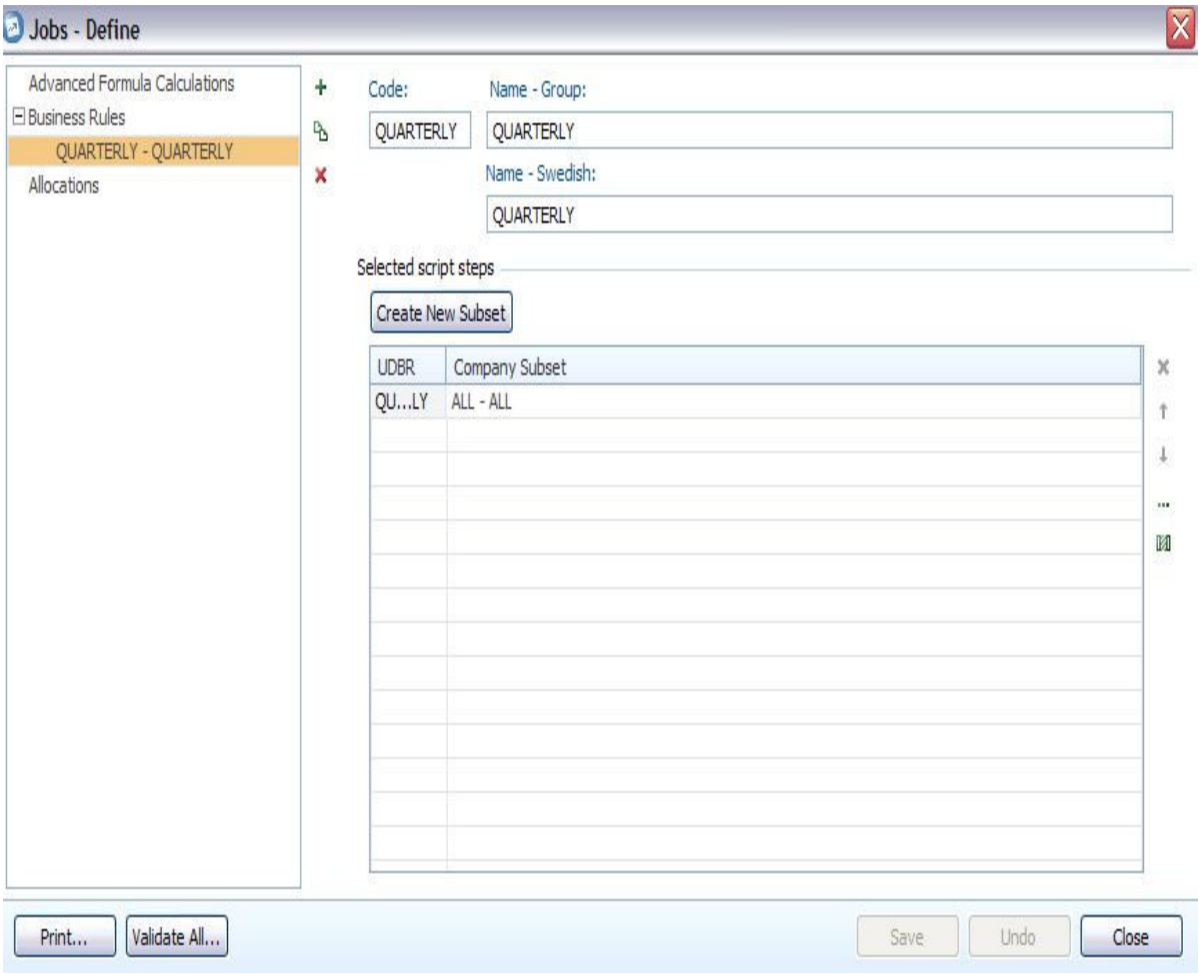


Figure 5 shows the defining of the UdBR jobs

Mapping of the UdBR job

- Go in maintain >jobs>mapping table
- Select "add a new row" to create the mapping
- Select the consolidation type and the actuality for which the business rule job must be run.
- Select also at which consolidation step you want to run the UdBR.

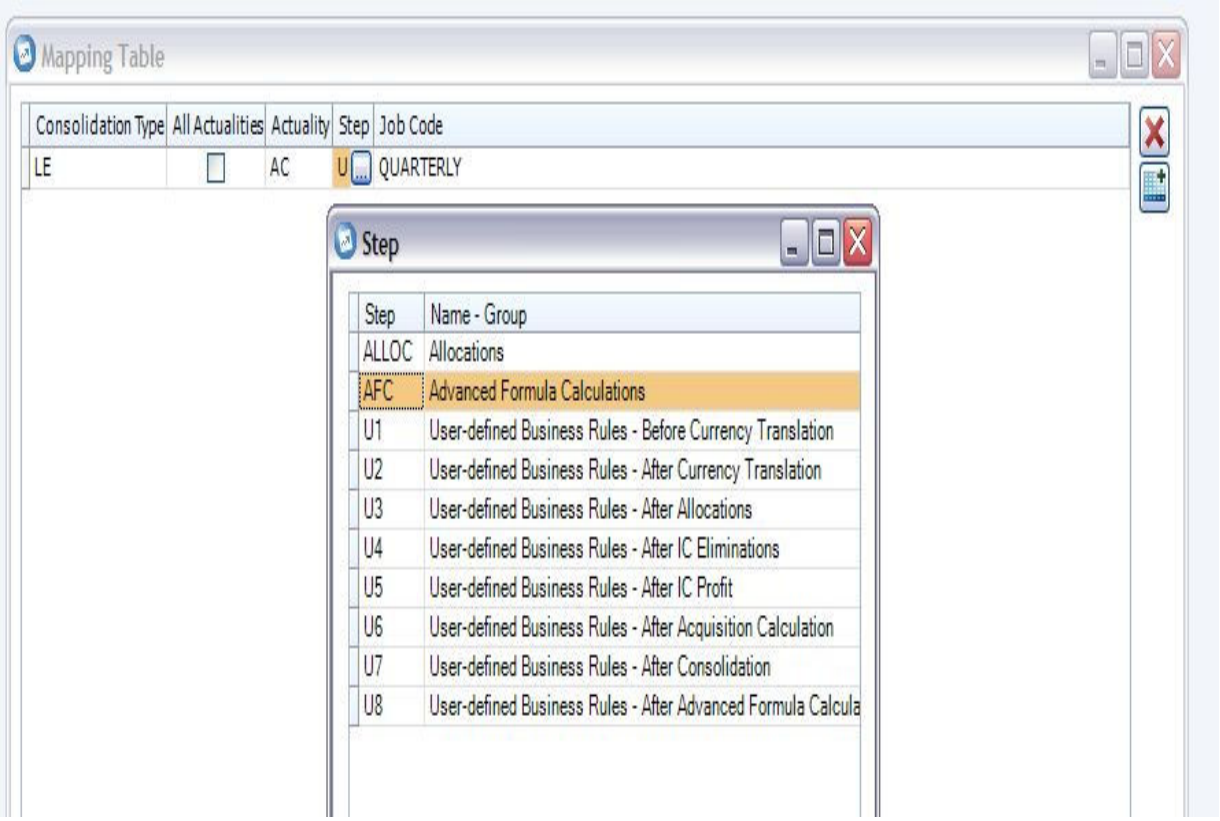


Figure 6 shows mapping table

Running of the Business rules

The business rules can be run directly in the menu maintain>business rules (more used for test purposes)

The business rules can be run via the **menu group>consolidate by steps>UdBR**

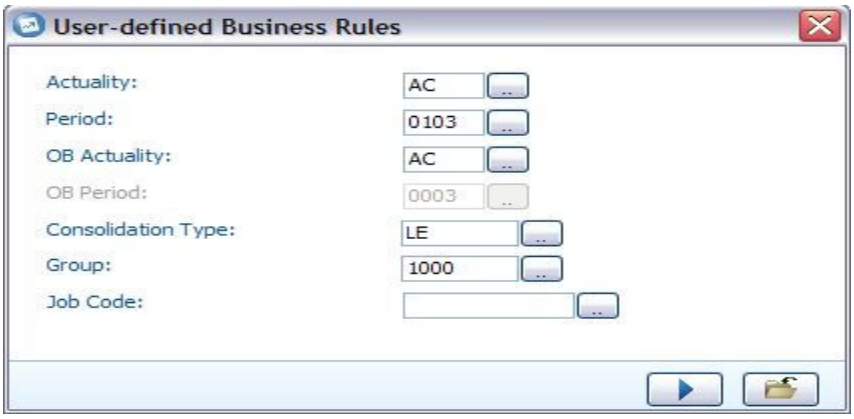


Figure 7 shows the parameters for consolidation by steps

The job code must be selected. The business rules can be run via the menu group>consolidate with status

By default the UdBR are part of the consolidation with status and the business rule job mapped to the consolidation type and actuality will be run

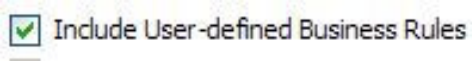


Figure 8 shows inclusion of UdBR as part the consolidation by steps

Python code

Python is a general-purpose, high-level programming language whose design philosophy emphasizes code readability.Python claims to combine "remarkable power with very clear syntax",and its standard library is large and comprehensive.

Narrative

It is best practice to narrate your code explaining what each section is doing and the reason.

To narrate your code prefix with a '#'

Start document:

Author

Date

Updates

Purpose

Variables

Reason

Use

Through the code

Amendments

Code: UDBR01

Name - Group: Move Data

Name - English: Move Data

Script:

```
#The purpose of this script is to copy the value from one account (SourceAccount)
#and move it to another account (TargetAccount)
#On copy the script multiplies the value by the multiplier
#This script was created by James Melville
#Created 2nd July 2012

#Define variables used in the script.
#Define the accounts we want to move data from and to.
SourceAccount = 'P10030'
TargetAccount = 'U10030'

#Define the multiplier we will multiply the value by.
Multiplier = 1.2

#Main part of the script
#Create the source values by pulling data from Controller
source1 = Creator.createSource()
source1.setAccount(SourceAccount)
source1.loadData()

#Now create the Target Values
target1= Creator.createTarget()
target1.setAccount(TargetAccount)
target1.addDataWithFactor(source1, 1.2)

#Save the values back to Controller
target1.mergeAndSaveData()
```

Figure 9 shows sample of UdBR code

Print

Sending items to print can check the present status of items and can assist with monitoring the status of the process and error checking

To print an item prefixes the item with the command 'Print'

See source file called: **PYTH01 – Print Method (2)**

Variables

Variables are elements/factors that are used within the code

Variables are elements/factors that are likely to change during the code.

Defining variables within Python is easy simply specify the name then give a '=' and then specify the value i.e.

myName = 'James Melville'

This defines the variable myName and gives it a value of 'James Melville'

Please refer to file called: **PYTH02 –Variable.txt**

String commands – concatenate

Join two string items to make one. Two string commands can be joined using a '+' command

Please refer to file called: **PYTH03 –concatenate.txt**

String Commands – Convert to string

Sometime the text you want to use could be interpreted by the system as a number i.e. '0'. In these cases we need to be explicit and use the str() command i.e. str(3) converts the number into a string representation of that number.

Please refer to file name: **PYTH04 –string method.txt**

String Commands – Convert to int

Sometime the numeric item you want to use could be interpreted by the system as a text value

In these cases we need to be explicit and use the int() command

i.e. int(myYear) converts the string within the variable into a numeric value representation of that number.

Please refer to file name: PYTH05 –int method.txt

Logical statements – If command

Some times you want to execute an area of code to see a specific criterion is met.

Python is a visual logical code i.e. end if statements are not needed it knows when the statement is completed by the indentation on the page.

If statement uses a comparison and then completes the comparison with a colon ':'. Then the code in the rows below and indented is executed if the value is correct.

Comparisons:

< less than

<= less than or equal to

> greater than

>= greater than or equal to

== equal

!= not equal

<> another way to say not equal

Please refer to file name: **PYTH06 – if statement.txt**

Logical statements – If else command

If commands can also do else and else if commands.

Else if is stated as 'elif' and followed by the comparison and then a colon ':'

i.e: elif myText == 'Big Company1':

The lines below indented one more space are the commands executed when this statement is true.

Else is stated as 'Else:' no comparison is required. If this is correct the rows below that are indented are executed.

Please refer to file name: **PYTH07 – if else statement.txt**

Logical statements – While statements

While statements will loop through whilst the criteria is met.

While statement starts with while, then has the comparison then the ':'

If the while statement is still valid it will execute the code indented below it.

Comparisons:

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal
!=	not equal
<>	another way to say not equal

Please refer to file named: PYTH08 – while statement.txt

Logical statements – for statements

For statements loop through items in a list/range

For statements start with for specifies the column/count then 'in' the list/range variable followed by the ':' i.e. for group in parents:

If the for statement is still valid it will execute the code indented below it.

Please refer to file named: **PYTH09 – for statement.txt**

Defining Functions

Functions are areas of code that can be reused by other parts of the program. Its aim is to reduce duplication and reduce maintenance overhead.

Def statements define functions and the syntax is 'Def' followed by function name, then brackets and any variables being passed and finally the ':' i.e. def printmynumber(num):

Functions will end once the indentation after it returns to normal or when a return command is used.

Return commands returns to the source section that called the function a final result.

Functions need to be defined in an early part of the code than the part that calls it.

Please refer to file named: **PYTH10 – define a function (2).txt**

UdBR creation and process

UdBR Creation

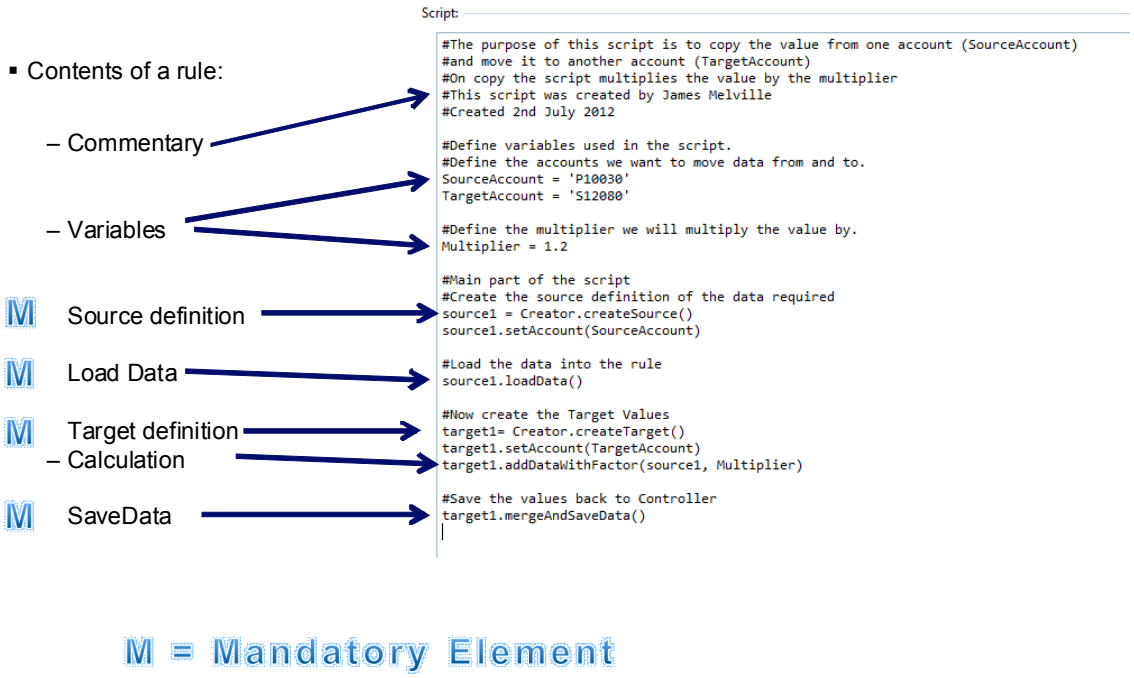


Figure 10 shows the flow of UdBr rules

The source & Load

A source for an UdBR must include at least the following parts:

The creation of the source with the following format: *XXXX=Creator.createSource()* , where

XXXX is the source name. All source functions must start with this name.

All source functions in the script must start with the source name followed by a "full stop" *.* and then the function.

Example: source1 = Creator.createSource()

The definition of the source thanks to functions (setAccount in the example below)

For example the source must contain an account code (P10030 in the example below).

Example: source1.setAccount('P10030')

A load data function that includes the name of the source (to load data from the Controller period tables)

Example: source1.loadPeriodData()

The default values :

By default if not set in the rule, the following values will be used for the source:

Source company: Same as in the job definition.

Current period: Same as used when running the consolidation.

Current actuality: Same as used when running the consolidation.

Current consolidation type: Same as used when running the consolidation (and if the UdBR is mapped to the consolidation type)

Manual journal type source: REPO

Automatic journal type source: BASE

Journal number source: blank

For the target if no specifications are set, it will take the source definition

The Target & Save:

A target definition for an UdBR must include at least the following parts:

The creation of the source with the following format: *ZZZZ=Creator.createTarget()* , where

ZZZZ is the target name. All source functions must start with this name.

All source functions in the script must start with the source name followed by a "full stop" and then the function.

Example: Result = Creator.createTarget()

The definition of the source thanks to functions (setAccount in the example below) .Example the target must contain an account code (U10030 in the example below).

Example: Result.setAccount('U10030')

A mergeAndSaveData function at the end of the definition, that includes the name of the target (to load the result into Controller)

Example: Result.mergeAndSaveData()

The calculation:

An optional calculation. For example in the target account I want to see the result of the division of source1 by source2 multiply by 100

Example: Result.addDataWithFactor(source1, 1)

Result.divideByTarget(source2, 1)

Result.multiplyByConstant(100)

The Pre-defined objects:

The following pre-defined objects are available to use in Source and Target functions instead of, or in addition to, specific values:

currentPeriod

obPeriod

currentCompany

currentActuality

obActuality

currentStructure

CompStruc

AccountModel

PeriodModel

Btype

Etype

Company

Writer

UDBR Process

Creation of the source definition

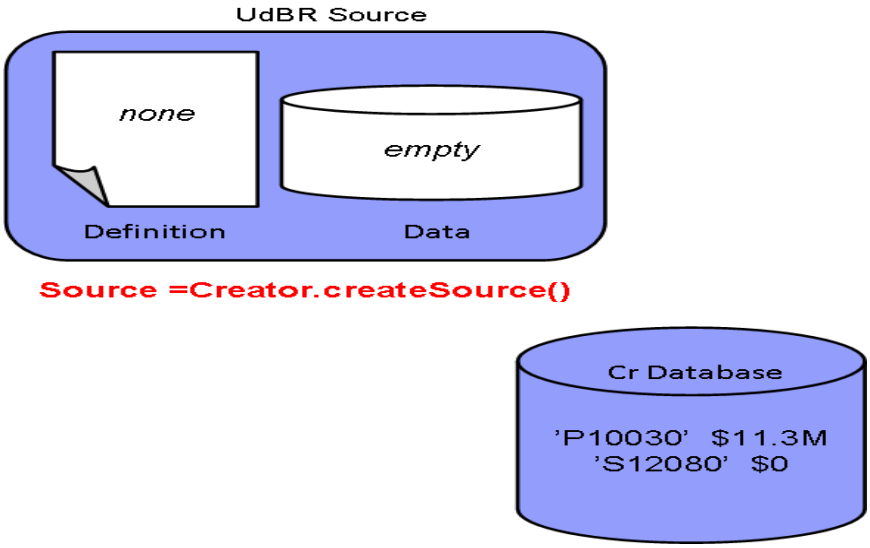


Figure 11 shows the creation of the source flow

Setting of the account and other dim in the source-definition

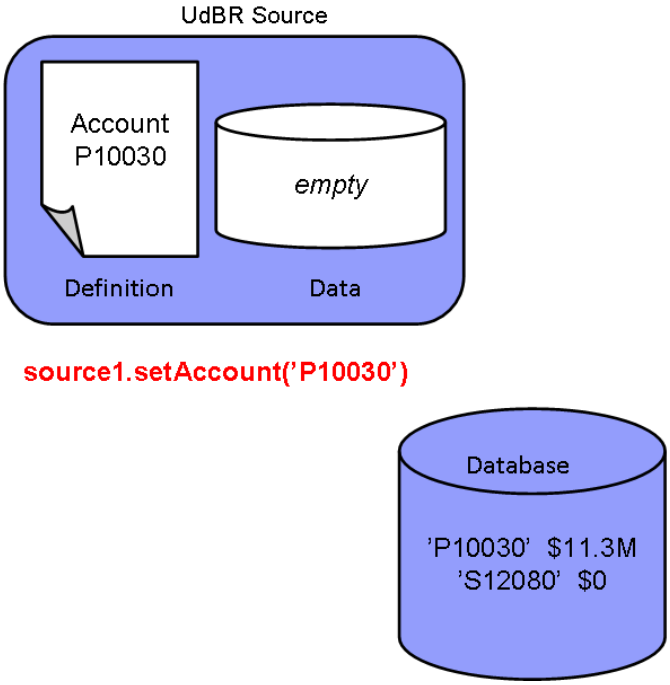


Figure 13 shows the source account for UdBR

Loading of data into the source:

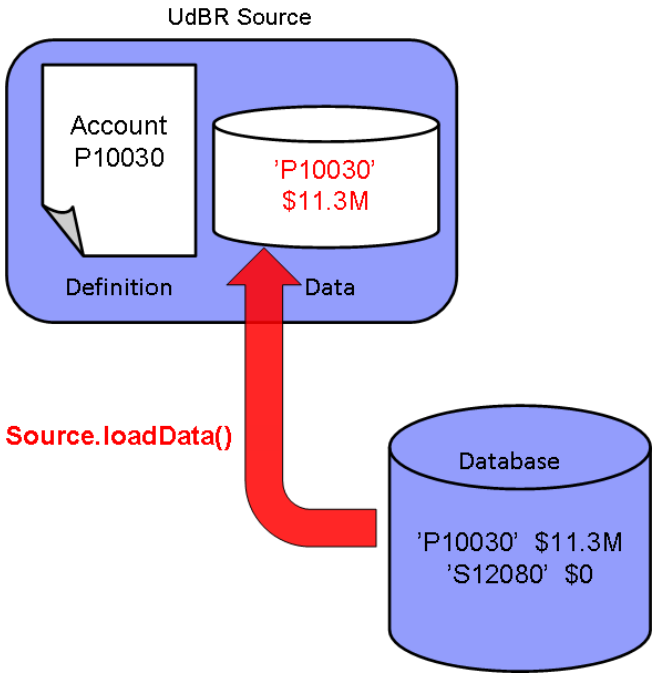


Figure 14 shows source date being pulled from the database

Definition of the target

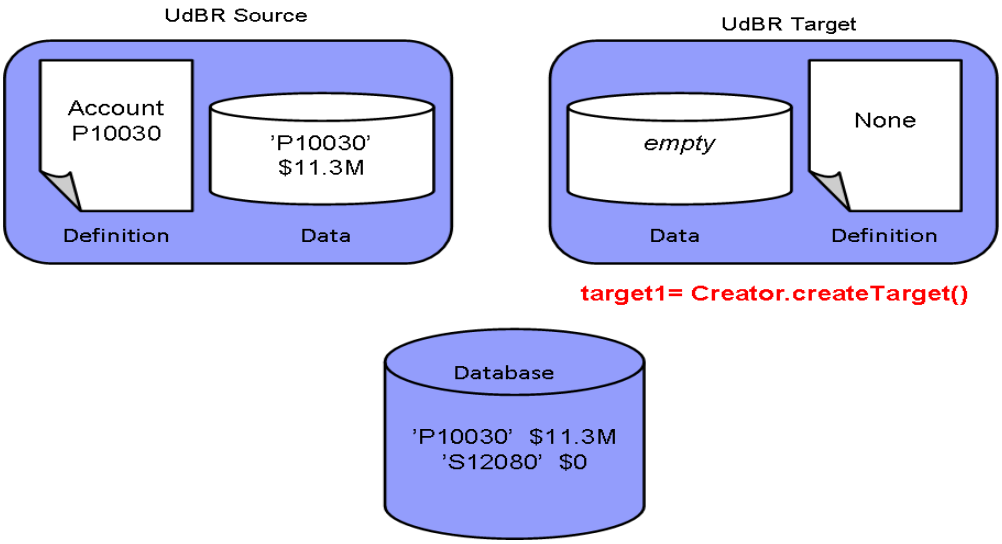


Figure 15 shows the target account definition flow

Definition of the account, dimension, … we want to send the data to:

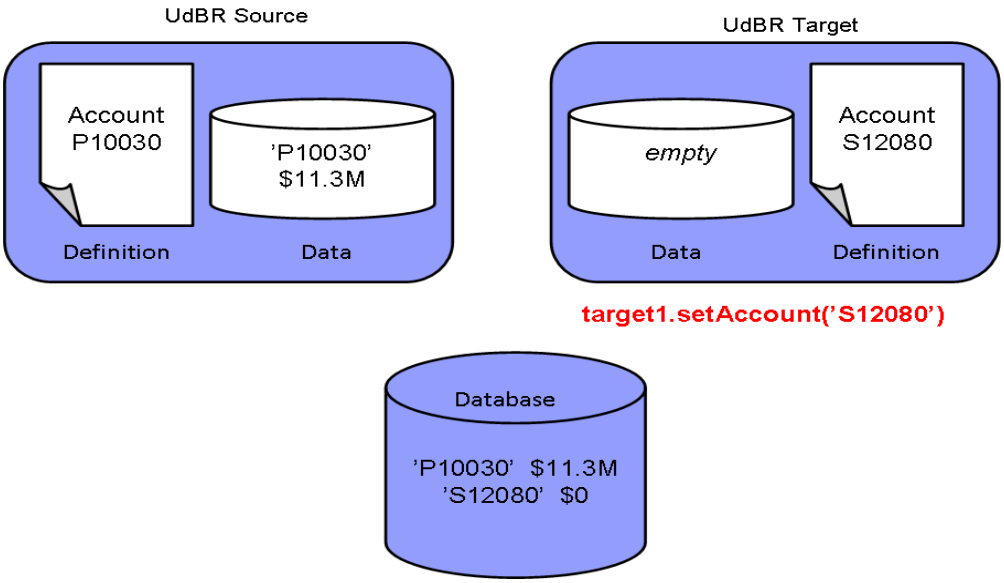
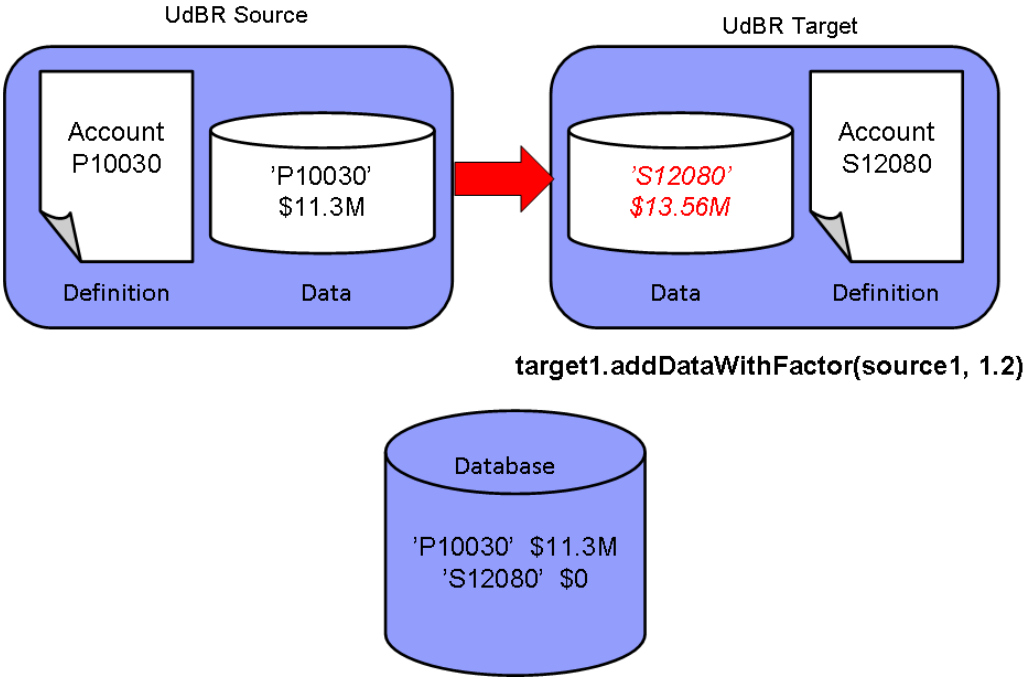


Figure 16 shows source and target accounts flow to and from database

Calculation and loading of data into the target



Loading of data into controller

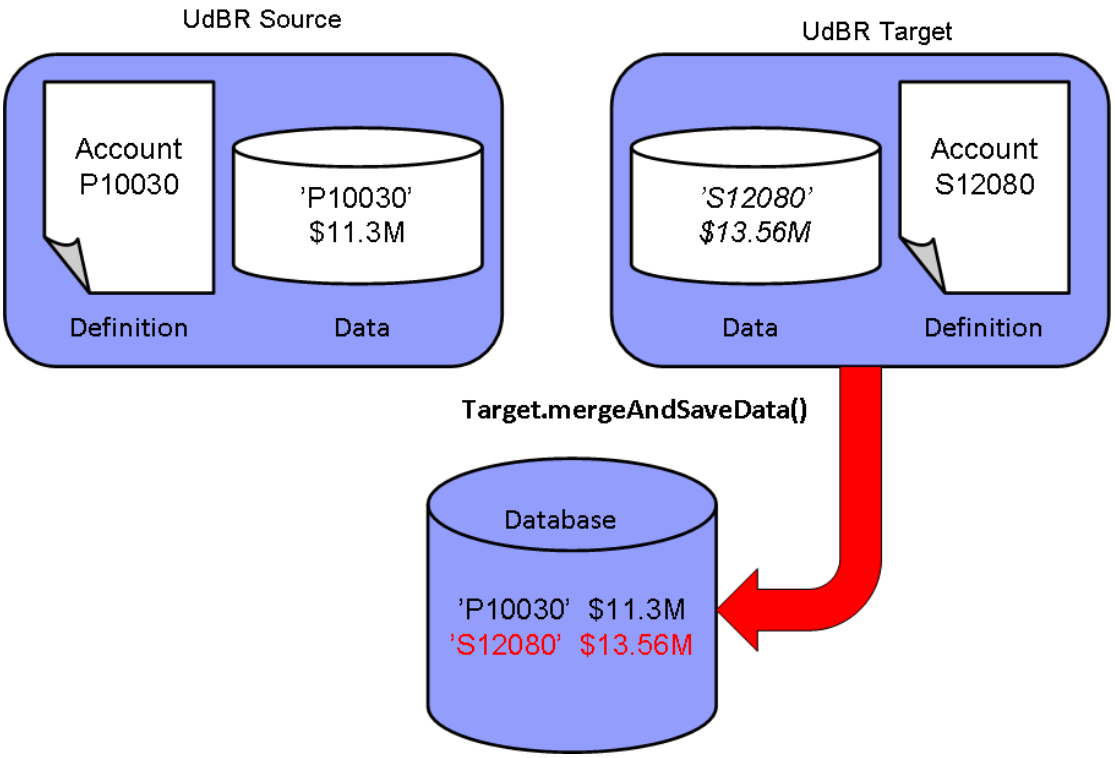


Figure 13 shows the value for both source and target stored in the database

UdBR Functions

Run time functions

Run Time Functions – Company information

When the consolidation is activated it will cycle through the periods selected and then within each period it will cycle through all the companies within the company subset. To retrieve the current company being worked on use the 'currentCompany' command.

In addition further information can be pulled back from the following commands:

currentCompany.getCode()

currentCompany.getName()

currentCompany.getCurrency()

currentCompany.getCompanyType()

(See attached file: **CCRF01 – current company.txt**)

Run Time Functions – Current Structure

currentStructure pulls the structure that is being consolidated.

currentStructure.getCode() will bring back just the code.

currentStructure.getName() will bring back the name.

(See attached file: **CCRF02 – current structure.txt**)

Run Time Functions – Current Period Information

A consolidation could be run for a single or multiple periods within the same year. UDBR when activated will cycle through all the periods within the consolidation process. To pull the present period being consolidated use currentPeriod.

In addition further information can be pulled back from the following commands:

currentPeriod.getActuality()

currentPeriod.getYear()

currentPeriod.getPeriod()

See file named: **PYTH11 – currentPeriod (2).txt**

Run Time Functions – Opening Period Information

A consolidation could be run for a single or multiple periods within the same year. UDBR when activated will cycle through all the periods within the consolidation process. To pull the opening period used by the submission being consolidated use obPeriod.

In addition further information can be pulled back from the following commands:

obPeriod.getActuality()

obPeriod.getYear()

obPeriod.getPeriod()

See file named: **PYTH14 – obPeriod.txt**

Run Time Functions – Actuality Information

CurrentActuality pulls the current actuality code.

obActuality pulls the opening actuality code specified in the submission being used for the consolidation.

See file named: **PYTH12 – currentActuality.txt**

See file named: **PYTH15 – obActuality.txt**

Run Time Functions – Company Structure

CompStruc object is used to pull ownership information direct from a company connection.

getParentCompanies – parameters (company, period). It will retrieve the group code

See file named: **PYTH16 - CompStruc.getParentCompanies.txt**

Consolidation Information

getConsolidationMethod – parameters (group, company, period)

getDeltaMinorityShare – parameters (group, company, period)

getDirectOwnershipPercentage – parameters (group, company, period)

getMinorityShare – parameters (group, company, period)

getOwnershipPercentage – parameters (group, company, period)

See file named: **PYTH16B - CompStruc.getParentCompanies.txt**

getGroupCurrencies – parameters (group, company, period). It will retrieve the group currency.

See file named: **PYTH16C - CompStruc.getParentCompanies.txt**

Run Time Functions – Account model

AccountModel.getAccountsInSummation pulls all leave level accounts that roll up to the summation account stated into a list.

Accounts can be circulated through using a for command.

Accounts retrieved could be at a lower level than directly below the summation account. i.e. Could be several levels below the mentioned account. i.e.

Account 1 & 2 sum up to 3

Accounts 4 & 5 sum up to 6

Accounts 3, 6, & 7 sum up to 8

If account 8 is specified 1, 2, 4, 5 & 7 will be returned and summation accounts 3 & 6 will be excluded.

Source data is pulled per account so this can be a critical model to pull more information from the source.

Account functions that pull further information are:

account.getCode()

account.getName()

account.getAccountType()

See file named: **PYTH17 - AccountModel.txt**

Run Time Functions – Period Model:

PeriodModel.getPeriodsSinceStartOfFiscalYear pulls all the periods prior to the period being specified into a list.

Accounts can be circulated through using a for command.

Source data is pulled per period so this can be a critical model to pull more information from the source.

Period functions that pull further information are:

period.getActuality()

period.getYear()

period.getPeriod()

Please refer to file named: **PYTH18 - PeriodModel.txt**

Source functions

These functions are used to define the source. All values are entered in brackets after the function name. The source part in the script must always start with the creator function *Creator.createSource().*

'set' functions filter data in the controller database from the load process

'groupBy' functions replaced previous values with stated values

'load', 'addData' & 'clear' functions load or clear data into the source area

```
#Define the multiplier we will multiply the value by.
Multiplier = 1.2

#Main part of the script
#Create the source definition of the data required
source1 = Creator.createSource()
source1.setAccount(SourceAccount)

#Load the data into the rule
source1.loadData()
```

Source Functions – Set Functions

Source Set Functions filter data extracted from Controller.

Source Set Functions are used on source definitions and are activated when loading data from controller.

Set Function	Argument	Example	Default
setPeriod	Period	source.setPeriod('1112')	Current Period
setPeract	Period + Actuality	source.setPeract('1112AC')	Current Period & Actuality
setRelativePeriod	Relative Period starting with either '+', '-', 'N' or 'P'	source.setRelativePeriod('-1')	'+0'
setAccount	Account Code	source.setAccount('P10030')	N/A - Must be Specified
setCompany	Company Code	source.setCompany('1102')	Current Company
setGroup	Group Code	source.setGroup('1100')	Current Company
setCompanyStructure	Company Structure Code	source.setCompanyStructure('LE')	Current Structure
setJournalNumber	Journal Number	source.setJournalNumber('0')	All Applicable to Journal Types
setBType	Manual Journal Type Code	source.setBType('IS')	All Applicable to Closing Version
setClosingVersion	Closing Version Code	source.setClosingVersion('IFRS')	REPO
setEType	Automatic Journal Type Code	source.setEType('35')	All Aplicable to Contribution version
setContributionVersion	Contribution Version Code	source.setContributionVersion('ALL')	BASE
setCurrency	Currency Code	source.setCurrency('USD')	*All*

Figure 14 shows UdBr functions and their descriptions

setPeriod

This function specifies the period for which data should be retrieved in the database.

Argument: Period code or an object defined in the script.

Example: source1. setPeriod ('1112') if not set will take the current period

setPeract

This function specifies the period and the actuality for which data should be retrieved in the database.

Argument: Period and actuality code or an object defined in the script.

Example: source1. setPeract ('1112AC') if not set will take the current period and actuality

setRelativePeriod

With this function, instead of specifying a specific period, as with the **setPeriod** function, you can retrieve data from a period relative to the current period.

Syntax: This uses the N112-notation:

The first character must be on of "N", "P", "+", "-"

If the first character is either "+" or "-" it must be followed by the number of periods relative to current period. E.g. If the current period is "1007AC" then the relative period "+1" is "1008AC"

If the first character is either "N" or "P" it must be followed by the number of years relative to current year and then the period number in that year. E.g. If the current period is "1007AC" then the relative period "N112" is "0912AC"

Example: Calculate quarterly values :

Source1.setPeriod (currentPeriod)

Source1.loadData ()

Source2. setRelativePeriod ('-3')

Source2. loadData ()

Source1. addDataWithFactor(source2, -1)

setAccount

This function sets the account for which data should be retrieved in the database. If a summation account is set then the aggregated value will be retrieved (not the details).

Argument: Account code.

Example: source1.setAccount('8999')

setCompany

This function specifies the company for which data should be retrieved in the database.

Note that the function retrieves data for one specific company.

Argument: Company code or an object defined in the script.

Example: source1. setCompany ('COGNOSHQ')

setCompanyStructure

This function sets the consolidation type (field ktypkonc) to be selected when loading the source.

You will use it only if you want to select automatic journals linked to a specific consolidation type

Argument: Consolidation type code or an object defined in the script.

Example: source1. setCompanyStructure ('LE')

setGroup

This function sets the group (field ktypkonc) to be selected when loading the source.

You will use it only if you want to select automatic journals linked to a specific group

Argument: Group code.

Example: source1. setGroup ('COGNOSGROUP ')

setJournalNumber

This function sets the journal number for which data should be retrieved in the database.

If not specified it will take all journal numbers

Argument: Journal number.

Example: source1. setJournalNumber ('0')

setBType

This function sets the manual journal type for which data should be retrieved in the database

Used if the manual journal type source must be other than Repo (set by default) and if only one specific journal must be retrieved (if not use set ClosingVersion instead)

Argument: Manual journal type code.

Example: source1.setBType('L')

setClosingVersion

This function sets the closing version for which data should be retrieved in the database.

Used if the closing version source must be other than REPO.

Argument: Closing version code.

Example: source1.setClosingVersion('LEGAL')

setEType

This function sets the automatic journal type for which data should be retrieved in the database

Used if the automatic journal type source must be other than BASE.

Argument: Automatic journal type code.

Example: source1.setEType('35')

setContributionVersion

This function sets the contribution version for which data should be retrieved in the database.

Used if the contribution version source must be other than BASE.

Argument: Automatic journal type code.

Example: source1. setContributionVersion('ALL')

setCurrency

This function sets the currency to be selected when loading the source.

If not used it will take all currencies

Argument: Currency code or an object defined in the script.

Example: source1. setCurrency ('USD')

Source Functions – Group by Functions

groupByBType

This function replaces all manual journal types in the source with the given manual journal type.

Argument: Manual journal type code.

This is done temporarily. The source transactions in the database are not changed by this function. If several transactions have the same manual journal type, and all other dimensions are the same, they are aggregated into one transaction.

Example: groupByBtype (BT)

	Source:	Btype	Etype	Amount
		Repo		1
		B1		2
		B2	E1	4
=>		BT		3
		BT	E1	4

groupByEType

This function replaces all the automatic journal type in the source with the given automatic journal type.

Argument: Automatic journal type code.

This is done temporarily. The source transactions in the database are not changed by this function. If several transactions have the same automatic journal type, and all other dimensions are the same, they are aggregated into one transaction.

Example: groupByEtype (E2)

Source:	Btype	Etype	Amount
	Repo		1
	B1		2
	B2	E1	4

=>	Repo	E2	1
	B1	E2	2
	B2	E2	4

Other Group by functions are:

groupByCounterCompany

groupByExtDim

groupByJournalNum

groupByOriginCompany

Source functions – Load Data:

loadData

This function loads the data from the database according to settings.

Example: source1. loadData ()

loadPeriodData

This function loads the periodic data, which will calculate the monthly value.

Use the current month and the previous month to calculate the value for the month. If it is the first month of the year, it will return the YTD value. It will check automatically what's the first month via the general configuration.

Note: This function does not work for weekly actualities or calculations of quarterly periods (use relative period instead)

clearData

This function clears the data of the source

This is done temporarily during the processing. The source transactions in the database are not cleared by this function. It can be used when you want to clear an intermediate calculation when doing a loop.

addDataWithFactor

With this function, data is added from another source. It includes two arguments: 1) A given source to add data from and 2) a factor to multiply all data with. A typical situation is to multiply with +1 or -1 to change a sign to positive and negative.

Argument: Source and factor.

Example: There are two transactions in source 1: car 2 and bike 3. There are two transactions in

source 2: truck 4 and mc 5.

The function is: Source1.addDataWithFactor (Source2,-2) ->

The result in Source1 will be car 2, bike 3, truck -8 and mc -10.

Source functions example:

Script:

```
for period in periodsSinceStartOfFiscalYear:
        periodNumber = period.getPeriod()
        print '  Present Period is: ' + str(periodNumber)

        # FETCH ALL GROUPS FOR THE CURRENT COMPANY FOR CURRENT AND PREVIOUS PERIOD
        groups = CompStruc.getParentCompanies(currentCompany, period)

        #  LOOP OVER GROUPS AND PULL OUT ANY STRUCTURE CHANGES
        for group in groups:
                print "        Group: " + group.getCode()
                consolidationMethod = CompStruc.getConsolidationMethod(group, currentCompany, period)

                directOwnedPercentage  = CompStruc.getDirectOwnershipPercentage(group, currentCompany, period)
                minorityPercentage = 1.0 - directOwnedPercentage

                structure = currentStructure.getCode()

                # FILTER ON COMPANIES THAT ARE PURCHASED OWNED.
                if consolidationMethod == "P" :
                        print '           This company uses the purchase methodology.'
                        print '           The present Ownership % is: ' + str(directOwnedPercentage)

                        # LOOP OVER GROUP CURRENCIES FOR CURRENT COMPANY
                        groupCurrencies = CompStruc.getGroupCurrencies(currentCompany)
                        for currency in groupCurrencies:
                                print "            Currency: " + currency.getCurrencyCode()

                        #Define the source data
                        source = Creator.createSource()
                        source.setPeriod(period)
                        source.setCompany(currentCompany)
                        source.setAccount(NetProfit)
                        source.setClosingVersion(ClosingVersion)
                        source.setContributionVersion(ContributionVersion)
                        source.setCurrency(currency)
                        source.loadPeriodData()

                        print source
```

Figure 15 show the UdBr script

Target functions

These functions are used to define the target. All values are entered in brackets after the function name. The source part in the script must always start with the creator function *Creator.createTarget().*

'set' functions sets the value of items when transferring them from source

'Calculation' items to add , multiply and divide values.

Save functions to save data back to Controller.

Target functions – Set Functions

Target Set Functions are used on target definitions and are activated when loading data from source.

Target Set Functions replace values in the source rows with new values. i.e. The target.setAccount will replace the existing account code from the source with a new one when placing it in to the target.

Target Set Functions do not filter data.

```
#Main part of the script
#Create the source definition of the data required
source1 = Creator.createSource()
source1.setAccount(SourceAccount)

#Load the data into the rule
source1.loadData()

#Now create the Target Values
target1= Creator.createTarget()
target1.setAccount(TargetAccount)
target1.addDataWithFactor(source1, Multiplier)

#Save the values back to Controller
target1.mergeAndSaveData()
```

Figure 17 shows UdBr Target rule

Set Function	Argument	Example	Default
setPeriod	Period	target.setPeriod('1112')	source period
setPeract	Period + Actuality	target.setPeract('1112AC')	source peract
setAccount	Account Code	target.setAccount('S12080')	Source Account Code
setExtDim	Dimension Code	target.setExtDim1('USA') or target.setExtDim2('1020') target.setExtDim3('XXXX') or target.setExtDim4('YYYY')	SourceDimension
setCompany	Company Code	target.setCompany('1102')	Source Company Code
setGroup	Group Code	target.setGroup('1100')	Source Group Code
setCompanyStructure	Company Structure Code	target.setCompanyStructure('LE')	Source Company Structure
setJournalNumber	Journal Number	target.setJournalNumber('0')	Source Journal Number
setBType	Manual Journal Type Code	target.setBType('IS')	Source Btype
setEType	Automatic Journal Type Code	target.setEType('35')	Source Etype
setCurrency	Currency Code	target.setCurrency('USD')	source currency
setTransferAmount	*None*	target.setTransferAmount()	N/A
setTransferCurrency	Currency Code	target.setTransferCurrency('USD')	N/A
setAmountToConstant	*Value*	target.setAmountToConstant(100)	source Value

Figure 18 show UdBR target functions

setPeriod

This function specifies the period to be used when saving the target. If not set it will take the source Period.

Argument: Period code or an object defined in the script.

Example:target1.setPeriod(currentPeriod)

setPeract

This function specifies the period and the actuality to be used when saving the target. If not set it will take the source Period and actuality

Argument: Period and actuality code or an object defined in the script.

Example: target1. setPeract ('1112AC')

setAccount

This function specifies the account where data should be stored in the database. If not set it will take the source Account

Argument: Account code.

Example:target1.setAccount('200000')

setExtDim

This function specifies the extended dimension 1 member on which the target amounts should be saved. This function will replace the original extended dimension 1 member.

It is possible to include several setExtDim1 functions, without having to reset the extended dimension

The level of the given extended dimension 1 member must comply with the level definition of the account. If not set it will take the source Dimension member

Argument: Extended dimension number 1-4 and extended dimension code.

Example: target1.setExtDim1 ('Car'), setExtDim2 ('Belgium')

setCompany

This function specifies the company to be used when saving the target.If not set it will take the source Company

Argument: Company code or an object defined in the script.

Example:target1.setCompany('4300')

setGroup

This function specifies the group (the ktypkonc field) to be used when saving the target.

If not set it will take the source Ktypkonc

Argument: Group code.

Example:target1.setGroup('4000')

setCompanyStructure

This function specifies the consolidation type (the ktypkonc field) to be used when saving the target.If not set it will take the source Ktypkonc

Argument: Consolidation type code or an object (currentStructure) defined in the script.

Example:target1.setCompanyStructure('LE')

setJournalNumber

This function specifies the journal number where data should be stored in the database. If not set it will take the source Journal number

Argument: Journal number.

Example:target1.setJournalNumber('0')

setBType

This function specifies the manual journal type where data should be stored in the database.If not set it will take the source Btype.

Argument: Manual journal type code.

Example : target1.setBType('L')

setEType

This function specifies the automatic journal type where data should be stored in the database.If not set it will take the source Etype

Argument: Automatic journal type code.

Example : target1.setEType('E1')

If you want to book the result of the business rule in an Automatic Journal, you must:

Set the automatic journal: setEType(' ')

Set the Journal number: setJournalNumber(' ')

Set the consolidation type (to get the consolidation type in the field ktypkonc):setCompanyStructure(currentStructure) if you want to book the automatic journals for several consolidation type or setCompanyStructure(' ') if you want to book it for a specific consolidation type

Load the result in Controller with the function saveDataWithJournalHeaders()

setCurrency

This function specifies the currency to be used when saving the target. If not set it will take the source Currency

Argument: Currency code or an object defined in the script.

Example:target1.setCurrency('USD')

setTransferAmount

This function will store the result in the transaction amount field.

Example : target1.setTransferAmount()

setTransferCurrency

This function specifies the transaction currency for the target amounts.If not set it will take the source transaction currency

Argument: Currency code.

Example : target1.setTransferCurrency ('USD')

setAmountToConstant

This function books as amount in the target the specified constant. This can be useful when wishing to divide a source value by a constant.

Argument: Constant

Example:target1.setAmountToConstant(100)

resetExtDim

With this function, the previously specified extended dimension member is reset.

Argument: Extended dimension number 1-4 number.

Target functions - Calculations:

addDataWithFactor

With this function, data is added from a source. It includes two commands: 1) Add data from a given source and 2) a factor to multiply all data with. A typical situation is to multiply with +1 or -1 to change a sign to positive or negative.

Argument: Source function and factor.

Example: Result.addDataWithFactor(source1, 1)

addSingleValue

This function adds a given value to the target.

Example: *Target.addSingleValue ('car',' ',' ',' ')* results in a new account with ext.dim1 = 'car'

clearData

This is done temporarily during the processing.

The source transactions in the database are not cleared by this function.It can be used when you want to clear an intermediate calculation when doing a loop.

divideByTarget

This function specifies what the target should be divided by.

Argument: Source function and source function, or target function and target function.

Example: divideByTarget (Source) means that the target is divided by the source.

The numerator and denominator need to be specified in the same way, that is, the same extended dimension 1, extended dimension 2, and so on. Transactions that do not match will not be included.

E.G: target1.divideByTarget(target2)

This means that target1 is the numerator and target2 is the denominator the result is stored in target1.

multiplyByConstant

With this function, all amounts will be multiplied with the given constant.

Example: mulitplyByConstant(100) will result in amounts 100 times higher.

Target functions – Save To Controller:

mergeAndSaveData

This function compares what is already in the database and the target, adds the difference (merge) and saves the total in the database.

A summation is done if there is a transaction in both the database and the target. If there is a transaction in the database that does not exist in the target, the transaction in the database will be deleted.

Must be used only if the booking is done in Repo/Base.

saveDataWithJournalHeaders

This function saves the data and generates relevant journal headers for the table XBJH .

Must be used when the booking is made in a journal.

In the description of the journal you will retrieve the name of the business rule.

Target functions example :

Script:

```
#print '           The present Ownership % is: ' + str(directOwnedPercentage)

# LOOP OVER GROUP CURRENCIES FOR CURRENT COMPANY
groupCurrencies = CompStruc.getGroupCurrencies(currentCompany)
for currency in groupCurrencies:
        #print "            Currency: " + currency.getCurrencyCode()

        #Define the source data
        source = Creator.createSource()
        source.setPeriod(period)
        source.setCompany(currentCompany)
        source.setAccount(NetProfit)
        source.setClosingVersion(ClosingVersion)
        source.setContributionVersion(ContributionVersion)
        source.setCurrency(currency)
        source.loadPeriodData()
        #print source

        #Define a Target Definition to be used later.
        def makeAdjustment(account, factor):
                target = Creator.createTarget()
                target.setPeriod(currentPeriod)
                target.setCompany(currentCompany)
                target.setAccount(account)
                target.setEType(AutoJournType)
                target.setJournalNumber(period.getPeriod())
                target.setCurrency(currency)
                target.setCompanyStructure(currentStructure)
                target.addDataWithFactor(source, factor)
                target.saveDataWithJournalHeaders()
                #print target

        #Minority Interest Element Posted To P&L
        makeAdjustment(MinorityInterestPL, -minorityPercentage)

        #Minority Interest Element Posted to BS
        makeAdjustment(MinorityInterestBS, -minorityPercentage)
```

Figure 19 shows UdBR target function sample script

Filter objects:

Filter objects are used if you want to keep only some of the loaded and/or calculated data.

These objects are python language function objects that are passed to the source and target functions "removeData" and "retainData".

First the code of the dim must be specified in a function as in the filter the dim code can not be typed directly .Except to filter a journal number , in this case the journal number can be typed directly

Note that in this first release there are no possibilities of filtering on extended dimensions

Note that the filter object must be typed one tab further.

Example 1: Remove all transactions at the source containing a journal number "0"

```
def XXXXX (transaction):
        return transaction.containsJournal(0)
source.removeData (XXXXX)
```

Example 2: Keep all transactions at the target containing an AJT 99

```
etyp99 = EType.forCode('99')
def XXXX(transaction):
        return transaction.containsEType(etyp99) and transaction.containsJournal(0)
target.retainData(XXXX)
```

List of filter objects:

```
boolean containsPeriod(period)
boolean isCompanyStructureDefined()
boolean containsCompanyStructure(structure)
```

boolean isGroupDefined()

boolean containsGroup(group)

boolean isCompanyDefined()

boolean containsCompany(company)

boolean containsAccount(account)

boolean isBTypeDefined()

boolean containsBType(btype)

boolean containsEType(etype)

boolean containsJournal()

boolean containsCurrency(currency)

Writer functions:

printSourceData

This function displays the source data when you press the **Run Script** button.

Argument: Source, an object defined in the script or created object.

Example: #Writer.printSourceData(source1)

printSourceDefinition

This function displays the source definition when you press the **Run Script** button.

Argument: Source, an object defined in the script or created object.

Example: #Writer.printSourceDefinition(source1)

printTargetData

This function displays the target data when you press the **Run Script** button.

Argument: Target, an object defined in the script or created object.

Example: #Writer.printTargetData(source1)

UdBR limitations

- It doesn't work on multi owned companies (companies linked to several subgroups)
- It's not possible to filter on extended dimensions
- It is not possible to do calculations on source and target numeric values

UdBR validation

- The best way to find an error in the UdBR is to run it in the menu Business rules on only one company (create a subset with only this company)
- In case of errors it will indicate the line and the reason
- Use print commands to ascertain potential errors
- To retrieve the line make a copy/paste in excel
- Then correct the error in the business rule
- Then rerun the UdBR until you don't get anymore error messages

UdBR examples

Split an amount by using a fix percentage.

In the example scripted below, we will remove the amount on the account 600000 and transfer 75% to the account 650000 and 25% to the account 680000.

Creation of the data source:

Source=Creator.createSource ()

Source.setAccount ('600000')

Source. setXXXX () must list here all the other source dim like company, period,…

Source.loadData ()

Remove the value of the source account and store the result in the same account:

Target1=Creator.createTarget ()

Target1.setBType('L') if you want to book the result in a manual journal "L"

Target1.setJournalNumber('101') if you want to book the result in a manual journal

Target1. setXXXX () must list here all the other target dim that are not the same as the source dim

Target1.addDataWithFactor (source,-1) to remove the amount of the account 600000

Target1.mergeAndSaveData ()

Target1.saveDataWithJournalHeaders () if you want to book the result in a manual journal

Store 75% of the value of the source account to a specific account:

Target2=Creator.createTarget ()

Target2.setAccount('650000')

Target2.setBType('L') if you want to book the result in a manual journal "L"

Target2.setJournalNumber('101') if you want to book the result in a manual journal

Target2. setXXXX () must list here all the other target dim that are not the same as the source dim

Target2.addDataWithFactor (source,0.75) to add 75% of the account 600000 to the account 650000

Target2.mergeAndSaveData ()

Target2.saveDataWithJournalHeaders ()if you want to book the result in a manual journal

Store 25% of the value of the source account to a specific account:

Target3=Creator.createTarget ()

Target3.setAccount('680000')

Target3.setBType('L') if you want to book the result in a manual journal "L"

Target3.setJournalNumber('101') if you want to book the result in a manual journal

Target3. setXXXX () must list here all the other target dim that are not the same as the source dim

Target3.addDataWithFactor (source,0.25) to add 25% of the account 600000 to the account 680000

Target3.mergeAndSaveData ()

Target3.saveDataWithJournalHeaders () if you want to book the result in a manual journal

Calculate intercompany profit margin per counterpart

The margin is the result of the division of the monthly gross profit by the monthly gross revenue.

The result will be stored on a margin account, with intercompany code M. This margin account will then be used for the intercompany profit elimination.

EXAMPLE:

In the example scripted below, the gross profit is the sum of the accounts: 409000, 500100,500400, 500500.

The gross revenue is the account 409000.

All these accounts are intercompany accounts.

The intercompany profit margin is a statistical account called interPMargin with intercompany code M. This account can be analyzed by dimension.

FORMULA:

interPMargin = monthlyGrossProfit / monthlyGrossRevenue

Where: monthlyGrossProfit = ((409000 periodvalue) + (500100 periodvalue)+ (500400 periodvalue)+ (500500 periodvalue))

monthlyGrossRevenue = (409000 periodvalue)

Creation of the data sources:

source1 = Creator.createSource()

source1.setAccount('400900')

source1.loadPeriodData() to get monthly values

source2 = Creator.createSource()

source2.setAccount('500100')

source2.loadPeriodData()

source3 = Creator.createSource()

```
source3.setAccount('500400')

source3.loadPeriodData()

source4 = Creator.createSource()

source4.setAccount('500500')

source4.loadPeriodData()

# Calculation of the numerator, that is, the monthly gross profit:

monthlyGrossProfit = Creator.createTarget()

monthlyGrossProfit.setAccount('INTERPMARGIND')

monthlyGrossProfit.addDataWithFactor(source1, 1)

monthlyGrossProfit.addDataWithFactor(source2, 1)

monthlyGrossProfit.addDataWithFactor(source3, 1)

monthlyGrossProfit.addDataWithFactor(source4, 1)

# Calculation of the denominator, that is, the monthly gross revenue:

monthlyGrossRevenue = Creator.createTarget()

monthlyGrossRevenue.setAccount('INTERPMARGIND')

monthlyGrossRevenue.addDataWithFactor(source1, 1)

# Calculation of the quotient, that is, the inter profit margin:

interPMargin = Creator.createTarget()

interPMargin.setAccount('INTERPMARGIND')

interPMargin.addDataWithFactor(monthlyGrossProfit, 1)

interPMargin.divideByTarget(monthlyGrossRevenue)
```

Additional script parts for adjusting data as intercompany margin account values:

interPMargin.multiplyByConstant(100) **to get the amount as a %**

interPMargin.setTransferAmount() the amount calculated must be booked in the Transaction amount field (by design for interprofitmargin accounts)

interPMargin.setTransferCurrency('PER') by design for interprofitmargin accounts

interPMargin.setAmountToConstant(1) must have 1 in the amount field as per design

interPMargin.mergeAndSaveData()

Calculate the depreciation of an asset

In the example scripted below, we will calculate 20% of depreciation on the account goodwill and book the result in an Automatic journal in the account depreciation goodwill.

#Account goodwill

main = Creator.createSource()

main.setAccount('1070')

main.loadData()

Account depreciation goodwill (20pc)

maindepreciation = Creator.createTarget()

maindepreciation.setAccount('1079')

maindepreciation.setEType('DG')

maindepreciation.setCompanyStructure(currentStructure) **to get the field ktypkonc filled in**

maindepreciation.setJournalNumber(1)

```
maindepreciation.addDataWithFactor(main, 0.20)
maindepreciation.saveDataWithJournalHeaders()

# account depreciation goodwill in Profit and loss
PandLdepreciation =Creator.createTarget()
PandLdepreciation.setAccount('7800')
PandLdepreciation.setEType('DG')
PandLdepreciation.setCompanyStructure(currentStructure)
PandLdepreciation.setJournalNumber(1)
PandLdepreciation.addDataWithFactor(main, 0.20)
PandLdepreciation.saveDataWithJournalHeaders()
# Update of the account net income in balance sheet
NettIncomebalsh =Creator.createTarget()
NettIncomebalsh.setAccount('2099')
NettIncomebalsh.setEType('DG')
NettIncomebalsh.setCompanyStructure(currentStructure)
NettIncomebalsh.setJournalNumber(1)
NettIncomebalsh.addDataWithFactor(main, -0.20)
NettIncomebalsh.saveDataWithJournalHeaders()

# Update of the specification of net income in balance sheet
```

```
NettIncomeMovem =Creator.createTarget()
NettIncomeMovem.setAccount('209965')
NettIncomeMovem.setEType('DG')
NettIncomeMovem.setCompanyStructure(currentStructure)
NettIncomeMovem.setJournalNumber(1)
NettIncomeMovem.addDataWithFactor(main, -0.20)
NettIncomeMovem.saveDataWithJournalHeaders()
print "End of booking of depreciation on goodwill for company/group: " + currentCompany.getCode()
```

Keep only positive or negative amounts

```
source = Creator.createSource()
source.setPeriod(currentPeriod)
source.setCompany('M50000')
source.setAccount('B310020')
source.setContributionVersion('ALL')
source.loadData()
Keep only positive amounts
targetPositive = Creator.createTarget()
targetPositive.setPeriod(currentPeriod)
targetPositive.setCompany('M50000')
targetPositive.setAccount('B310020')
```

```
targetPositive.addDataWithFactor(source, 1)

def positiveFilter(amount):
    if (amount.getAmount() > 0):
        return True
    else:
        return False

targetPositive.retainData(positiveFilter)
```

Keep only negative amounts

```
targetNegative = Creator.createTarget()
targetNegative.setPeriod(currentPeriod)
targetNegative.setCompany('M50000')
targetNegative.setAccount('B310020')

targetNegative.addDataWithFactor(source, 1)

def negativeFilter(amount):
    if (amount.getAmount() < 0):
```

```
            return True
        else:
            return False

targetNegative.retainData(negativeFilter)
```

YTD average based on number of spent days

See attached file called: **YTDAverage.txt**

QTD average based on number of spent days

See attached file called: **qtrToDateAverageRefined.txt**

Handling of structural changes during the year

See the specific proven practice document