BANNER

General Person Synchronization Guide

CONFIDENTIAL BUSINESS INFORMATION

****

This documentation is proprietary information of SunGard Higher Education and is not to be copied, reproduced, lent or disposed of, nor used for any purpose other than that for which it is specifically provided without the written permission of SGHE.

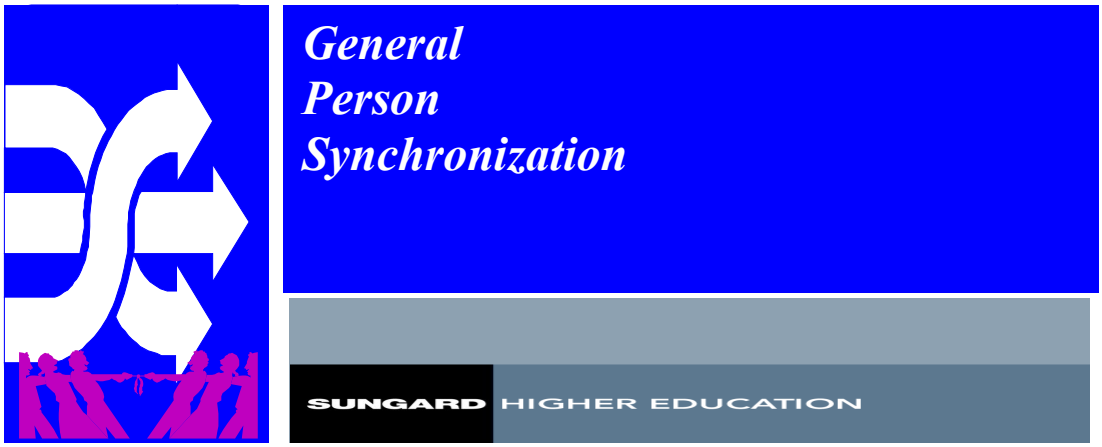
****

Prepared For: BANNER System Release 7.0/8.0
Prepared By: SunGard Higher Education
4 Country View Road
Malvern, PA 19355
Date Prepared: November 2008

Copyright 2008 SunGard Higher Education. All Rights Reserved.

Table of Contents

Introduction ***3***
Merge Functions ***8***
SPRIDEN ***9***
F_PREWP ***9***
F_PRE_LOOP_CM_API ***10***
F_CVT_CURCERR_CLEAN ***11***
F_PRE_LOOP_EXT_REFRESH ***12***
F_PRE_LOOP_IND_REFRESH ***13***
F_SUSPENSE_OVERRIDE ***14***
F_PRE_LOOP_IDEN_MERGE ***16***
F_WRAP_CURR_ID_FOR_NAME_CHG ***19***
F_WRAP_CURR_NAME_FOR_ID_CHG ***20***
SPBPERS ***21***
F_PRE_LOOP_WRAPPER ***21***
F_PRE_LOOP_EXT_REFRESH ***22***
F_PRE_LOOP_IND_REFRESH ***23***
F_ PRE_LOOP_PERS_MERGE ***24***
SPRADDR ***26***
F_PRE_LOOP_WRAPADR ***26***
F_PRE_LOOP_EXT_REFRESH ***27***
F_PRE_LOOP_IND_REFRESH ***28***
F_ PRE_LOOP_ADDR_MERGE ***29***
Address heirarchy ***31***
Address Type Override by Merge String ***32***
SPRTELE ***33***
F_PRE_LOOP_WRAPTEL ***33***
F_PRE_LOOP_EXT_REFRESH ***34***
F_PRE_LOOP_IND_REFRESH ***35***
F_ PRE_LOOP_TELE_MERGE ***36***
GOREMAL ***38***
F_PRE_LOOP_WRAPEML ***38***
F_PRE_LOOP_EXT_REFRESH ***39***
F_PRE_LOOP_IND_REFRESH ***40***
F_ CVT_WRAP_GOREMAL_DISP_WEB ***41***
F_ PRE_LOOP_EMAL_MERGE ***42***
Implementation ***44***
SPRIDEN ***45***
SPBPERS ***49***
SPRADDR ***52***
SPRTELE ***56***
GOREMAL ***59***
GRID-OracleDirectories- External tables ***62***
Temp Tables Creation ***64***
Functions compilation ***64***
Indexes needed ***65***
APPENDIX ***67***
Big Bang Concept ***67***
Troubleshooting ***75***
UPD_STATUS meanings ***76***
Flow Chart ***79***



General Person merging of matched records
Implementation - Migration - Interfacing

While General Person data itself is not over complicated and conceptually it is easy to understand – developing the rules to identify when an incoming record pre-exists in Banner and… developing the rules on how to merge those incoming person/non-person records to those that pre-exist in Banner… can be challenging.

During a Banner Implementation, General Person data may or may not pre-exist in Banner for an individual (or corporation). This truly depends on which Banner systems are "Live" and where in the implementation cycle your particular system is going "Live". If you are not the very first system to be implemented OR you are implementing in a phased approach OR General Person data exists in the General Person Module then you must check first to see if the person/non-person you are loading already exists in the system.

Determining the rules by which a record set is deemed to pre-exist in Banner General Person ***has*** to be defined by each Institution. This document assumes that Banners Common Matching API was used to determine if the record set is a "Match" or "Suspense" (potential match) based on rules defined in Banner. However – any match method could be used as long as that method flags the record set as "Matched" or "Suspended" and loads the Error logging table in the same manner as the function that calls the Common Matching API does. This document is not focusing on defining HOW-TO find a match or suspense – but on the more complicated task of merging file data to pre-existing Banner records.

Determining the rules by which a record set can be merged to pre-existing Banner records can become fairly complex. Typically this then involved custom coding to perform the merge for each Institutions rules. True that this usually meant taking a program developed for some other client and modifying it based on the current clients needs – but this then meant that there was no real standardized approach between conversions, consultants and clients.

Issue: How then do we bridge the gap between flexibility/customization and standardization/consistency?

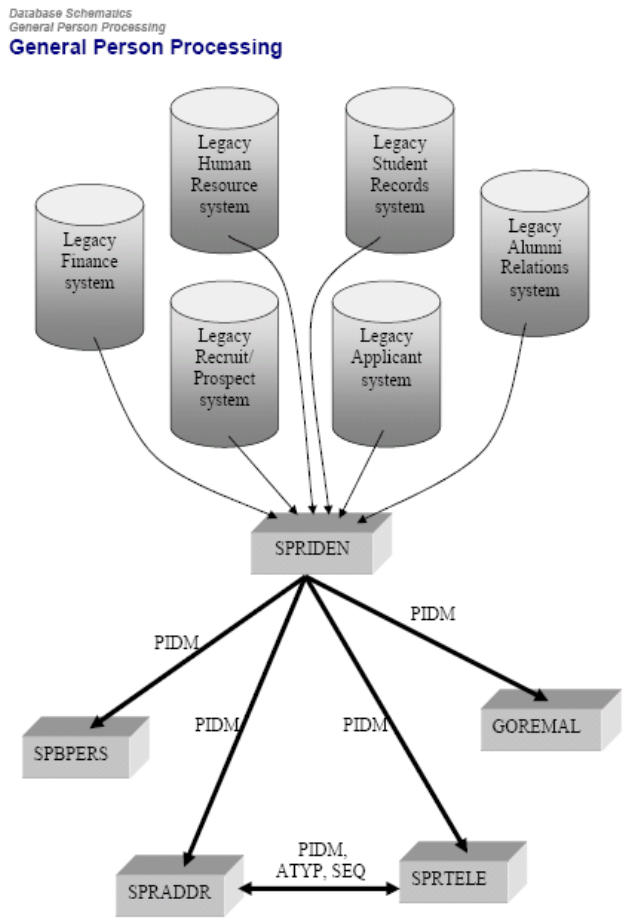
Assumptions:
Each *system* is converted one at a time even if doing in one fell swoop.

GPSynch is "INTO Banner only"… this is not a bidirectional synchronization. *Please engage SGHE for technical assistance in extraction of data from Banner for feeds to a legacy system.*

*** For "BIG BANG" General Person synchronization concepts – please see the "BIG BANG Concept" in the Appendix*

Resolution:
Welcome to the ***General Person Synchronization*** methodology!

What is ***GPSynch*** and what role does it play in the merging of matched record sets?



The ***GPSynch*** methodology uses a *Grid*; which is an Excel spreadsheet in which the client can define whether the incoming file data takes precedence over existing Banner data OR if the existing Banner data takes precedence over the incoming file data… a "Who wins?" methodology. Once the grid is completed (filled out) – it is saved as a comma delimited file and used as the source for an Oracle External table, which in turn is used by the merging functions, which will be discussed later.

Since generally we do not take such a simplistic approach to biographic data merging – File vs. Banner per system – we need flexibility while at the same time achieving consistency. We need to take into account questions like:

“Is the record set dealing with a Current Employee that is also a Current Student?”

“Is the record set dealing with a Current Vendor that is also a Current Student?”

“Is the record set dealing with a Current Vendor that is also a Constituent?”

“Is the record set dealing with a Constituent that is also a Current Student?”

“Is the record set dealing with a Current Employee that is also a Current Student who is a Current Vendor and a Constituent?”

“Is the record set dealing with a Former Employee that is also a Current Student who is a Current Vendor and a Constituent?”
Etc. etc. etc.

This is where ***GPSynch*** comes into play. The process uses a set of *codes* that represent the state of the record set in each of the four main systems in Banner: Finance, Human Resources, Student and Advancement.

NOTE: *Financial Aid is not included as that system does not convert General Person; that conversion occurred with the Student System. If Financial Aid was converting General Person data then we would allow that data to fall into the Student System code set.*

The codes and their meaning are listed below:

Finance
CV – Current Vendor
FV – Former Vendor
NO – not in Finance

Human Resources
CE – Current Employee
FE – Former Employee
NO – Not in Human Resources

Student
CS – Current Student
FS – Former Student
NO – Not in Student

Advancement
CA – Current Alum/Constituent/Friend
NO – Not in Advancement

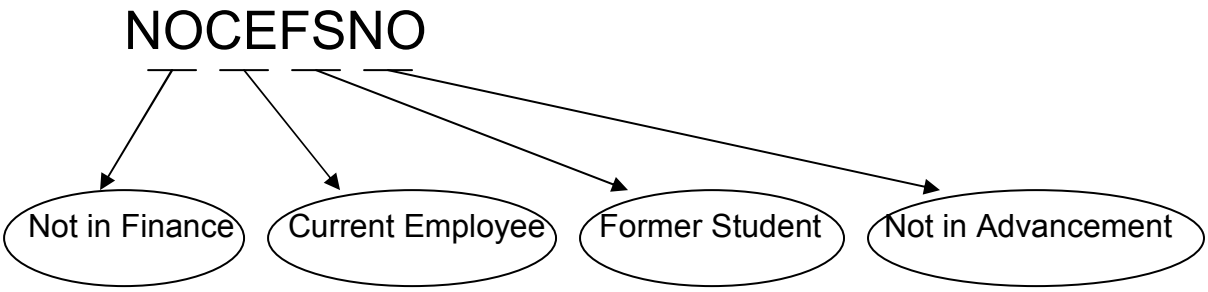
These codes are gathered from each system and then concatenated together to create an eight character “merge string”. This “merge string” is then used to read the WHO_WINS table (created from the WHO_WINS_EXT Oracle External table using the Grid spreadsheet) and returns the client defined value of either “BAN” – Banner wins
or “FIL” - File data wins.

> *For example*: The Institution has already gone live with Finance and HR but not Advancement and the file is an extraction from the Institutions Student system. The Institution develops a policy where pre-existing HR Banner records are always the most up-to-date if the record is attached to a

Current Employee – therefore Banner wins. However, if the record is attached to a Former Employee and you are bringing in a Current Student record then the file data from the Student system can update Banner.

We could have a record set that was found to have pre-existing data in Banner. Suppose our record set is from a Current Employee who once was a Student at the Institution and therefore pre-exists in Banner and has been extracted from the legacy Student system.

The merge string would be determined as follows:



The merge string always concatenates as follows:

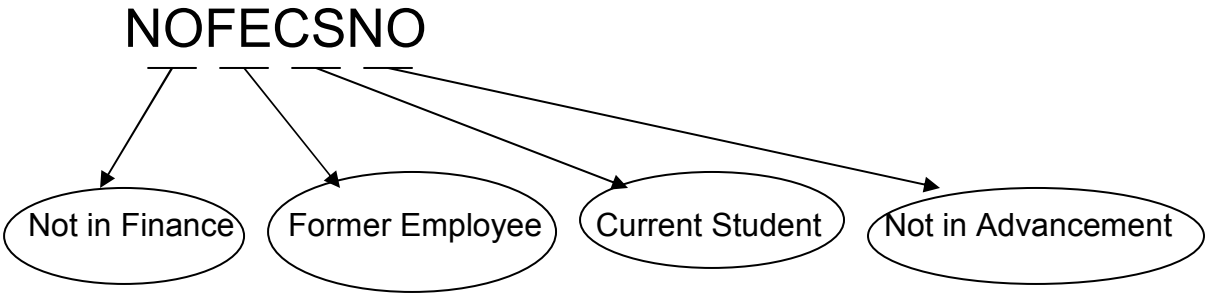
Finance || Human Resources || Student || Advancement

***NOTE*:** This order was chosen on purpose and is to represent the typical implementation model… Finance, then Human Resources, then Student, then Advancement. (Implementation order can differ; this is just the typical model order)

Using this merge string and the Banner column name, we can fetch the winner from the grid. The returned value depends on how the client defined the *winner* in the Grid. From our information above – the grid would return BAN – meaning Banner is more current and should remain so. We can not overwrite current data in Banner – but we can add our data as needed - name changes, other address types, etc.

Next, we could have a record set that was found to have pre-existing data in Banner. Suppose our record set is from a Former Employee who is now a Current Student at the Institution and therefore pre-exists in Banner and has been extracted from the legacy Student system.

The merge string would be determined as follows:



Using this merge string and the Banner column name, we can fetch the winner from the grid. The returned value depends on how the client defined the winner in the grid. From our information above – the grid would return FIL – meaning File data is more current and should update Banner. Later in this document we will talk about how each merge function works for each of the five core General Person tables – some things we update, some things we make inactive and load our record as new.

Once the merge string in derived, the merge string is then utilized by the merge functions to determine how to programmatically merge the data from the File with the data in Banner. This is done by reading the grid for the merge string and for the table/column you are merging.

The use of the merge string allows us to define a ***precedence*** of how the data should be merged. As in our example above, the status of our record set really determined what we are able to do to the pre-existing Banner records. The grid has 54 different merge string combinations that work with the columns that are found in the five core General Person tables: SPRIDEN, SPBPERS, SPRADDR, SPRTELE, GOREMAL. The grid can be updated as needed, as rules change, etc. allowing for a flexible program that does not require any code modifications(other than potential crosswalking). The merge functions utilize the Banner baseline API's for inserting and updating.

Say hello to flexibility and customization.

Meet standardization and consistency.

Next we will discuss the design and logic of each merge function and the custom columns needed for each temp table.

Merge Functions

This section provides a general discussion of the logic of each function. This section is NOT a step by step discussion of the code itself, that can be accomplished by reading the source code. Later in this document we will discuss the steps needed to implement the ***GPSynch*** methodology for your general person conversion.

There are 5 core functions in the ***GPSynch*** methodology; one for each of the core tables utilized in Common Matching: SPRIDEN, SPBPERS, SPRADDR, SPRTELE and GOREMAL.

Other functions exist to compliment the methodology or would be required in a non-merging General Person conversion.

SPRIDEN

F_PREWP.SQL

This function is utilized with the table SPRIDEN. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function actually houses calls to seven (7) other functions needed to process the SPRIDEN table using this methodology. Other custom functions could be added to this function if needed.

This function accepts four (4) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

Cmsc_code_in – this is the Common Matching code you plan on using. Short names are best like “CONV” or “STU”

records_in – this is the spriden_cvt_status records that you want to process, typically N or C flagged records.

Cur_jobid – this is the run number that will be stored in CURCERR.

This function then calls the following seven (7) functions:

F_PRE_LOOP_CM_API.SQL
F_CVT_CURCERR_CLEAN.SQL
F_PRE_LOOP_EXT_REFRESH.SQL
F_PRE_LOOP_IND_REFRESH.SQL
F_SUSPENSE_OVERRIDE.SQL
F_PRE_LOOP_IDEN_MERGE.SQL
F_WRAP_CURR_ID_FOR_NAME_CHG.SQL
F_WRAP_CURR_NAME_FOR_ID_CHG.SQL

These functions will be discussed next.

F_PRE_LOOP_CM_API.SQL

This function is utilized with the table SPRIDEN. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (spriden_convert.sql) disposition is 'C' (process_level = 'C') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts four (4) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

cmsc_code – this is the Common matching source rule name you wish to use

records_in – this is the spriden_cvt_status records that you want to process, typically N or C flagged records.

cur_jobid – this is the value that comes from the cubcnvt_sequence that corresponds to the run number of the main conversion program spriden_convert.sql

This function pulls record sets as a Cartesian product from SPRIDEN_CVT, SPBPERS_CVT, SPRADDR_CVT, SPRTELE_CVT, GOREMAL_CVT that have been flagged as "new" (spriden_cvt_status = 'N') in the temp table spriden_cvt. The function then uses the common matching API using the provided common matching source code to determine if the record pre-exist in Banner (spriden_cvt_status becomes 'M' for match) or is a suspicious potential match (spriden_cvt_status becomes 'S' for suspense). API messages for Match or Suspense records are stored in the error table CURCERR for later use by the suspense override and merging functions.

It is recommended that you review the source code of the function so that you understand the logic. ***Please note that this function is very large and is not viewable from the Converter Tool "view Function" form – use SQL Developer or similar tool. This function is actually a procedure in "disguise" as a function***

F_CVT_CURCERR_CLEAN.SQL

This function is utilized with the table SPRIDEN. As its name indicates, the function is intended to clean up error message from previous synchronizations. This function only truly launches when the conversion routine (spriden_convert.sql) disposition is 'C' (process_level = 'C') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" .

This function accepts one (1) parameter:

Perform_clean – baseline is delivered 'Y'.

This function performs the "clean" only if the parameter fed to the function is 'Y' (which is the delivered default) AND only if ALL records in SPRIDEN_CVT are new (spriden_cvt_status = 'N'); as this would indicate a new synchronization and we want to ensure previous synchronization rows do not exist in CURCERR. In this case… the function performs an EXECUTE IMMEDIATE of the statement "truncate table curcerr"

It is recommended that you review the source code of the function so that you understand the logic.

F_PRE_LOOP_EXT_REFESH.SQL

This function is utilized with the table SPRIDEN. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (spriden_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function refreshes the Oracle table WHO_WINS from the Oracle External Table WHO_WINS_EXT and the Oracle table SUSPENSE_OVERRIDE from the Oracle External Table SUSPENSE_OVERRIDE_EXT. This was done for two reasons: (1) Oracle External tables can not be indexed (2) to reduce I/O on the OS and stop large log files from being produced on the OS.

We want to use the Oracle External table method so that the Grid rules can update dynamically – reducing the need for extra steps to get the merge code to recognize rules changes over time.

It is recommended that you review the source code of the function so that you understand the logic.

F_PRE_LOOP_IND_REFESH.SQL

This function is utilized with the table SPRIDEN. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (spriden_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function ensures the necessary indexes exist for the methodology.

It is recommended that you review the source code of the function so that you understand the logic.

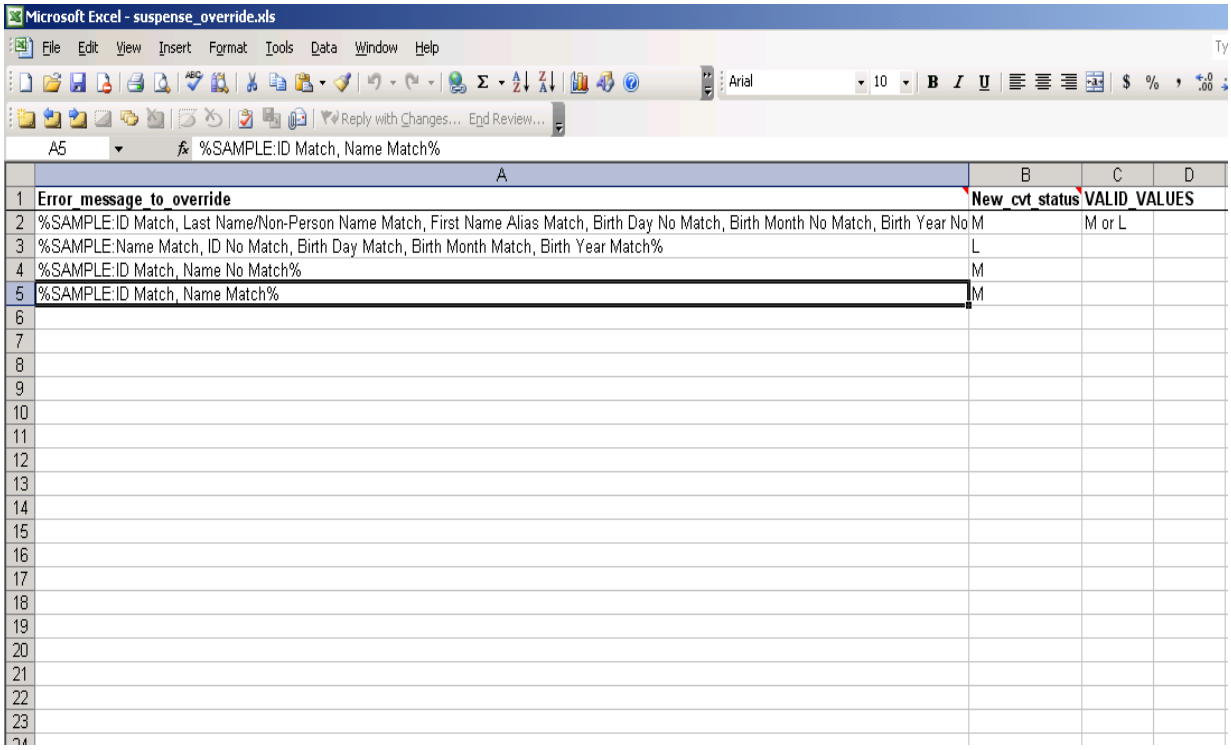
F_SUSPENSE_OVERRIDE.SQL

This function is utilized with the table SPRIDEN. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (spriden_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function retrieves API suspense error messages from an Oracle table SUSPENSE_OVERRIDE.



Error_message_to_override	New_cvt_status	VALID_VALUES
%SAMPLE:ID Match, Last Name/Non-Person Name Match, First Name Alias Match, Birth Day No Match, Birth Month No Match, Birth Year No	M	M or L
%SAMPLE:Name Match, ID No Match, Birth Day Match, Birth Month Match, Birth Year Match%	L	
%SAMPLE:ID Match, Name No Match%	M	
%SAMPLE:ID Match, Name Match%	M	

This message is then used to update the SPRIDEN_CVT_STATUS = ‘S’ for the record in the table SPRIDEN_CVT to have a new SPRIDEN_CVT_STATUS of either:

> SPRIDEN_CVT_STATUS = ‘M’
> override the suspended record to mean actually an exact match. The merge function will only pull ‘M’ flagged records for merging.
>
> SPRIDEN_CVT_STATUS = ‘L’
> override the suspended record to LOAD as a new record. We use L as the function that we use to call common matching would never pull records flagged with a SPRIDENCVT_STATUS = ‘L’, that function is designed to only pull raw or new records – meaning SPRIDEN_CVT_STATUS = ‘N’. So, your routines would have an L to C run now for the SPRIDEN_CONVERT.SQL program.

The function then takes appropriate action of cleaning the error logging table for these overrides. To maintain an audit, the messages are not removed. Overridden messages to ‘L’ will be prefixed with ‘OVR-‘. Overridden messages to ‘M’ are simply updated from ‘S’ to ‘M’ in curcerr.

It is recommended that you review the source code of the function so that you understand the logic.

F_PRE_LOOP_IDEN_MERGE.SQL

This function is utilized with the table SPRIDEN. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (spriden_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts three (3) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

system_converting – this is the legacy system you are converting. Valid values are: SIS, HRS, FRS, ADS

enrollment_term – This value is optional and not needed when the implementing legacy system is SIS and Admissions is not live. Other systems can provide this value to determine (1) a currently enrolled student by this cut off term and/or (2) a recent applicant in Admissions. ***However, when the implementing legacy system is SIS – the extraction utility of the legacy data must place the word "CURRENT" in the CONVERT_DATA_ORIGIN column of the temp table SPRIDEN_CVT for those ID's that the institution deems to be current students (each institution will have to define what a "current student" actually means). This is the only way we will be able to determine a "current student" as Banner Student is being implemented and is, for all intensive purposes, empty.***

This function only pulls record sets from SPRIDEN_CVT that have been flagged as "matched" by setting the spriden_cvt_status = 'M' in the temp table spriden_cvt. The function then uses baseline Banner functions to determine the record sets status in the Banner Systems (f_alumni_constituent_ind, f_alumni_organization_ind, f_alumni_friend_ind, f_finance_vendor_ind, f_payroll_employee_ind, f_student_enrollment_ind, f_student_admissions_ind).

NOTE: if you are converting all General Person from all Legacy Systems first and then synchronizing – see the appendix on "Big Bang Concept" for alternatives to the functions above

The value from each system is then concatenated together to create a "merge string". This merge string is then used to read the grid to determine if the Banner data takes precedence over the Legacy File data or vice-versa.

Microsoft Excel - who_wins.xls

A116 GOREMAL_PIDM

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	column_name	CVCECSCA	CVCECSNO	CVCEFSCA	CVCEFSNO	CVCENOCA	CVCENONO	CVFECSCA	CVFECSNO	CVFEFSCA	CVFEFSNO	CVFENOCA	CVFENONO
2	SPRIDEN_PIDM	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
3	SPRIDEN_ID	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
4	SPRIDEN_LAST_NAME	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
5	SPRIDEN_FIRST_NAME	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
6	SPRIDEN_MI	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
7	SPRIDEN_CHANGE_IND	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
8	SPRIDEN_ENTITY_IND	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
9	SPRIDEN_ACTIVITY_DATE	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
10	SPRIDEN_USER	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
11	SPRIDEN_ORIGIN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
12	SPRIDEN_SEARCH_LAST_NAME	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
13	SPRIDEN_SEARCH_FIRST_NAME	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
14	SPRIDEN_SEARCH_MI	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
15	SPRIDEN_SOUNDEX_LAST_NAME	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
16	SPRIDEN_SOUNDEX_FIRST_NAME	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
17	SPRIDEN_NTYP_CODE	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
18	SPRIDEN_CREATE_USER	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
19	SPRIDEN_CREATE_DATE	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
20	SPRIDEN_DATA_ORIGIN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
21	SPRIDEN_CREATE_FDMN_CODE	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
22	SPRIDEN_SURNAME_PREFIX	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
23	SPBPERS_PIDM	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
24	SPBPERS_SSN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
25	SPBPERS_BIRTH_DATE	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
26	SPBPERS_LGCY_CODE	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
27	SPBPERS_ETHN_CODE	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
28	SPBPERS_MRTL_CODE	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
29	SPBPERS_RELG_CODE	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
30	SPBPERS_SEX	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
31	SPBPERS_CONFID_IND	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
32	SPBPERS_DEAD_IND	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
33	SPBPERS_VETC_FILE_NUMBER	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
34	SPBPERS_LEGAL_NAME	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN
35	SPBPERS_PREF_FIRST_NAME	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN	BAN

who wins matrix / Sheet2 / Sheet3

Ready

Since SPRIDEN is a repeating table, we have the ability to add rows to this table. Therefore, when reading the grid, the function will only ever look at the values placed at the intersection of the SPRIDEN_PIDM row and column containing the derived merge string value. However, other columns for the table shown in **RED** are actionable as well.

The function then takes appropriate action of inserting new records and updating existing records based off the rule “learned” from the grid. Exactly matching data is flagged by placing the value ‘m’ in a custom column called SPRi-DEN_UPD_STATUS (custom columns will be discussed in the IMPLEMENTA-TION section, values used in the %_UPD_STATUS columns are documented in the appendix).

For example:

> You are implementing SIS. Human Resources and Finance have already gone live in Banner. You record was flagged by Common Matching to be a match to a pre-existing record in Banner. The record coming from SIS is for a Current Student and the record in Banner is for a Former Employee. Your institution has defined the grid such that a Current Student’s SIS data can update Human Resources data for a Former Employee.
>
> The merge_string = ‘NOFECSNO’ and the grid returned ‘FIL’. The function determines that the SIS data is different than the current record in Banner. The function then passes the current record from the SIS FILE record set to the p_update procedure of the API for SPRIDEN. The Banner record is set to a non-null change indicator value and the SIS File record becomes the current Banner record. Name and ID changes from SIS are loaded into

SPRIDEN if not exactly matching an existing change record. Depending on *how* the data is different from the pre-existing Banner data… both a name change record and an ID change record may be created. Further, if the Banner ID was a generated ID.. and the incoming record “won” as in this example; you can configure the SPRIDEN_ID column so that BAN is the winner for ID… for the over all table FIL is the winner. This would allow the retention of the generated Banner ID to be attached to the Current Banner record.. and the Legacy ID to become an alternate.

If you need to do any special data processing, crosswalking, etc. Then these calls need to be placed in the function code as the functions always read from the raw data columns of the temp table. This should be the only need to modify the delivered code – however, other customizations may be needed depending on your institution – though typically that is not the case.

It is recommended that you review the source code of the function so that you understand the logic. Currently the functions are designed using IF-THEN-ELSE-END IF which can become confusing.

F_WRAP_CURR_ID_FOR_NAME_CHG.SQL

This function is utilized with the table SPRIDEN. As its name indicates, the function is implemented as a Wrap Up function; meaning that this function will be called after the cursor loop in the conversion program. This function was designed to run when the conversion routine (spriden_convert.sql) disposition is 'C' (process_level = 'C') as a clean up routine after the N to C run. However, we get the same effect by calling it as a Pre Loop function on a C to I run.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function ensures that the Name change records being processed are associated to the ID that is (or is going to be) the current Banner ID. The API will not allow the name change to be loaded with an ID other than the current Banner ID (the ID associated with the spriden row where spriden_change_ind is null).

It is recommended that you review the source code of the function so that you understand the logic.

F_WRAP_CURR_NAME_FOR_ID_CHG.SQL

This function is utilized with the table SPRIDEN. As its name indicates, the function is implemented as a Wrap Up function; meaning that this function will be called after the cursor loop in the conversion program. This function was designed to run when the conversion routine (spriden_convert.sql) disposition is 'C' (process_level = 'C') as a clean up routine after the N to C run. However, we get the same effect by calling it as a Pre Loop function on a C to I run.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function ensures that the ID change records being processed are associated to the Name information that is (or is going to be) the current Banner Name. The API will not allow the ID change to be loaded with name information other than the current Banner record (the record associated with the spriden row where spriden_change_ind is null).

It is recommended that you review the source code of the function so that you understand the logic.

SPBPERS

F_PRE_LOOP_WRAPPER.SQL

This function is utilized with the table SPBPERS. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function actually houses calls to three (3) other functions needed to process the SPBPERS table using this methodology. Other custom functions could be added to this function if needed.

This function accepts two (2) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

system_converting – this is the legacy system you are converting. Valid values are: SIS, HRS, FRS, ADS

This function then calls the following three (3) functions:

F_PRE_LOOP_EXT_REFRESH.SQL
F_PRE_LOOP_IND_REFRESH.SQL
F_PRE_LOOP_PERS_MERGE.SQL

These functions will be discussed next.

F_PRE_LOOP_EXT_REFESH.SQL

This function is utilized with the table SPBPERS. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (spbpers_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function refreshes the Oracle table WHO_WINS from the Oracle External Table WHO_WINS_EXT and the Oracle table SUSPENSE_OVERRIDE from the Oracle External Table SUSPENSE_OVERRIDE_EXT. This was done for two reasons: (1) Oracle External tables can not be indexed (2) to reduce I/O on the OS and stop large log files from being produced on the OS.

We want to use the Oracle External table method so that the Grid rules can update dynamically – reducing the need for extra steps to get the merge code to recognize rules changes over time.

It is recommended that you review the source code of the function so that you understand the logic.

F_PRE_LOOP_IND_REFESH.SQL

This function is utilized with the table SPBPERS. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (spbpers_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function ensures the necessary indexes exist for the methodology.

It is recommended that you review the source code of the function so that you understand the logic.

F_PRE_LOOP_PERS_MERGE.SQL

This function is utilized with the table SPBPERS. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (spbpers_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts three (3) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

system_converting – this is the legacy system you are converting. Valid values are: SIS, HRS, FRS, ADS

enrollment_term – This value is **optional** and not needed when the implementing legacy system is SIS and Admissions is not live. Other systems can provide this value to determine (1) a currently enrolled student by this cut off term and/or (2) a recent applicant in Admissions. ***However, when the implementing legacy system is SIS – the extraction utility of the legacy data must place the word "CURRENT" in the CONVERT_DATA_ORIGIN column of the temp table SPRIDEN_CVT for those ID's that the institution deems to be current students (each institution will have to define what a "current student" actually means). This is the only way we will be able to determine a "current student" as Banner Student is being implemented and is, for all intensive purposes, empty.***

This function only pulls record sets from SPBPERS_CVT that have been flagged as "matched" by setting the spbpers_cvt_status = 'M' in the temp table spbpers_cvt. The function then fetches the Merge String from SPRIDEN_CVT that had already been derived from the merging of Matched SPRIDEN records. This merge string is then used to read the grid to determine if the Banner data takes precedence over the Legacy File data or vice-versa. Since SPBPERS is a base table, we DO NOT have the ability to add rows to this table. Therefore, when reading the grid, the function will look at the values placed at the intersection of the SPBPERS column name row and column containing the derived merge string value. This gives us the ability to specify appropriate action for each column in SPBPERS and not the table as a whole.

> For Example:
> We want to allow the Legacy File data to overwrite the Ethnicity if the record set is not a Current Employee, but we will never allow the overwrite of SSN if one exists in Banner.

We specify in the intersection of the appropriate columns (merge string value) for the row containing the column name SPBPERS_ETHN_CODE and place the code FIL.

We specify in the intersection of the appropriate columns (merge string value) for the row containing the column name SPBPERS_SSN and place the code BAN.

The function then takes appropriate action of inserting new records and updating existing records based off the rule “learned” from the grid. If no row exists for this record then one is created. If a row exists, then appropriate updates are made based off of the rule “learned” from the grid. Further, if a row exists in SPBPERS, but the column is empty – the function will always add the new data regardless of the value pulled from the grid.

If you need to do any special data processing, crosswalking, etc. Then these calls need to be placed in the function code as the functions always read from the raw data columns of the temp table.

It is recommended that you review the source code of the function so that you understand the logic. Currently the functions are designed using IF-THEN-ELSE-END IF which can become confusing.

SPRADDR

F_PRE_LOOP_WRAPADR.SQL

This function is utilized with the table SPRADDR. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function actually houses calls to three (3) other functions needed to process the SPRADDR table using this methodology. Other custom functions could be added to this function if needed.

This function accepts two (2) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

system_converting – this is the legacy system you are converting. Valid values are: SIS, HRS, FRS, ADS

This function then calls the following three (3) functions:

F_PRE_LOOP_EXT_REFRESH.SQL
F_PRE_LOOP_IND_REFRESH.SQL
F_PRE_LOOP_ADDR_MERGE.SQL

These functions will be discussed next.

F_PRE_LOOP_EXT_REFESH.SQL

This function is utilized with the table SPRADDR. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (spraddr_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function refreshes the Oracle table WHO_WINS from the Oracle External Table WHO_WINS_EXT and the Oracle table SUSPENSE_OVERRIDE from the Oracle External Table SUSPENSE_OVERRIDE_EXT. This was done for two reasons: (1) Oracle External tables can not be indexed (2) to reduce I/O on the OS and stop large log files from being produced on the OS.

We want to use the Oracle External table method so that the Grid rules can update dynamically – reducing the need for extra steps to get the merge code to recognize rules changes over time.

It is recommended that you review the source code of the function so that you understand the logic.

F_PRE_LOOP_IND_REFESH.SQL

This function is utilized with the table SPRADDR. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (spraddr_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function ensures the necessary indexes exist for the methodology.

It is recommended that you review the source code of the function so that you understand the logic.

F_PRE_LOOP_ADDR_MERGE.SQL

This function is utilized with the table SPRADDR. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (spraddr_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts four (4) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

system_converting – this is the legacy system you are converting. Valid values are: SIS, HRS, FRS, ADS

enrollment_term – This value is optional and not needed when the implementing legacy system is SIS and Admissions is not live. Other systems can provide this value to determine (1) a currently enrolled student by this cut off term and/or (2) a recent applicant in Admissions. ***However, when the implementing legacy system is SIS – the extraction utility of the legacy data must place the word "CURRENT" in the CONVERT_DATA_ORIGIN column of the temp table SPRIDEN_CVT for those ID's that the institution deems to be current students (each institution will have to define what a "current student" actually means). This is the only way we will be able to determine a "current student" as Banner Student is being implemented and is, for all intensive purposes, empty.***

override_date – this is a cut off date that is provided at runtime to allow the override of the WHO_WINS value of FIL to be forced to BAN. The intention is that if the address activity date in Banner is greater than this override date then Banner is acceptably newer and even if FIL is listed in the Grid – BAN should be used as we want to retain the Banner data as current.

This function only pulls record sets from SPRADDR_CVT that have been flagged as "matched" by setting the spraddr_cvt_status = 'M' in the temp table spraddr_cvt. The function then fetches the Merge String from SPRIDEN_CVT that had already been derived from the merging of Matched SPRIDEN records. This merge string is then used to read the grid to determine if the Banner data takes precedence over the Legacy File data or vice-versa. Since SPRADDR is a repeating table, we have the ability to add rows to this table. Therefore, when reading the grid, the function will only ever look at the values placed at the inter-

section of the SPRADDR_PIDM row and column containing the derived merge string value.

The function then takes appropriate action of inserting new records and updating existing records based off the rule “learned” from the grid. Exactly matching data is flagged by placing the value ‘m’ in a custom column called SPRADDR_UPD_STATUS (custom columns will be discussed in the IMPLEMENTATION section, values used in the %_UPD_STATUS columns are documented in the appendix).

If the address in Banner exactly matches the address from the FILE for the same address type but the address status is different; the function uses the rule “learned” from the grid and updates the address status appropriately (or does not update).

If the FILE data is not exactly matching but a match is found on address types alone, the function determines if the FILE address date range (SPRADDR_FROM_DATE and SPRADDR_TO_DATE) is outside of the defined range for the matching address type in Banner. If the FILE record is outside of the Banner address date range then the FILE address is loaded with the next available sequence number for the PIDM and ATYP. Keep in mind that the conversion routine for SPRTELE will need to fetch this new address sequence number to maintain the true link between the address and the telephone number (this will be discussed with the function for merging telephone numbers in SPRTELE).

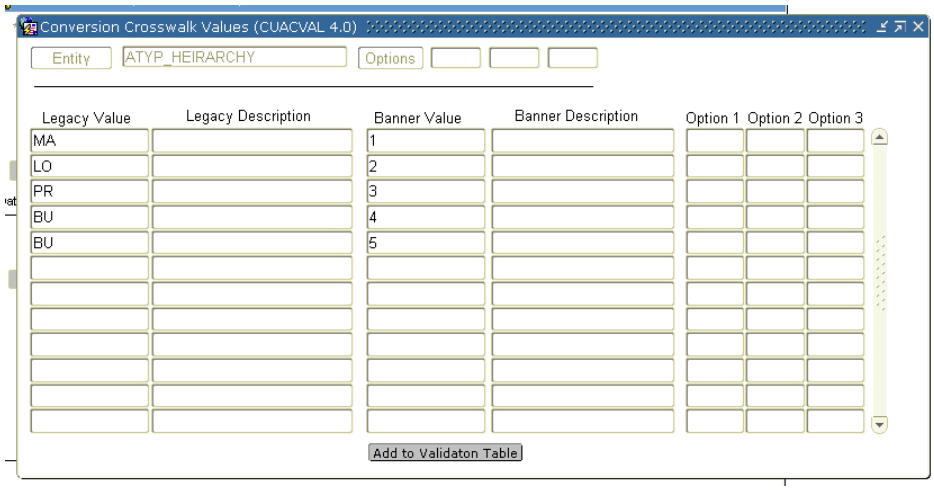
For example:

> You are implementing SIS. Human Resources and Finance have already gone live in Banner. You record was flagged by Common Matching to be a match to a pre-existing record in Banner. The record coming from SIS is for a Current Student and the record in Banner is for a Former Employee. Your institution has defined the grid such that a Current Student’s SIS data can update Human Resources data for a Former Employee.
>
> The merge_string = ‘NOFECSNO’ and the grid returned ‘FIL’. The function determines that the SIS data is different than the current record in Banner for the address type (SPRADDR_ATYP_CODE). The function does further work to determine if the FILE address date range (SPRADDR_FROM_DATE and SPRADDR_TO_DATE) is outside of the date range for the existing Banner address of the same address type. If the FILE record is found to be within the Banner record date range, the function then passes the current record from the SIS FILE record set to the p_update procedure of the API for SPRADDR. The Banner record is set to inactive by terminating the SPRADDR_TO_DATE to a value one day less than the SPRADDR_FROM_DATE of the FILE data and the FILE record is loaded using the next available sequence number. If the FILE record is found to be outside the Banner record date range, the function

then passes the current record from the SIS FILE record set to the p_create procedure of the API for SPRADDR. The FILE record is loaded using the next available sequence number.

“Active” Address hierarchy functionality

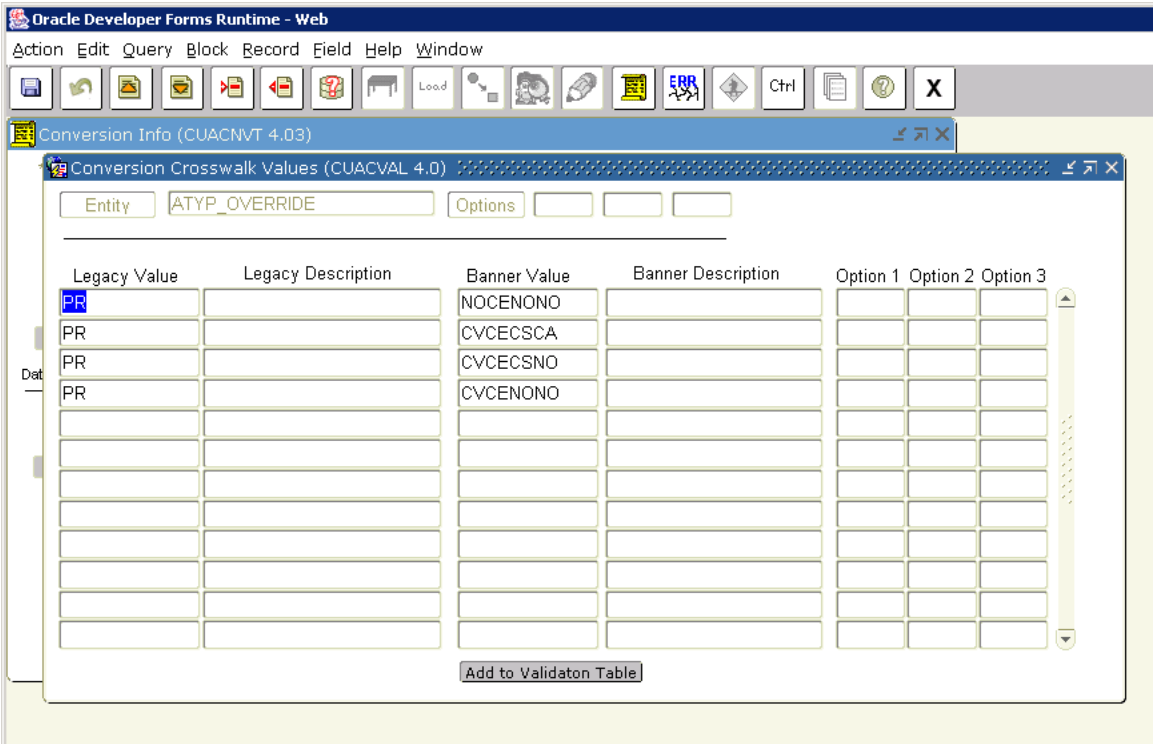
This function has the further capability of setting an “Active” hierarchy for the FILE data when your Legacy Address Type code crosswalk has multiple Legacy Address Type codes becoming one Address Type Code in Banner (i.e. LO and MA Address Types from Legacy will both become MA Banner Address Type codes). The issue is as follows: The Legacy record set for the ID has an active LO Legacy Address Type that is Active and also an MA Legacy Address Type that is Active. Both of the Legacy Address Type codes will become MA Banner Address Type codes. Banner can only have one Active Address Type for a date range at a time. We achieve this by creating a custom column on temp table SPRADDR_CVT called SPRADDR_ATYP_HEIRARCHY and developing a crosswalk called ATYP_HEIRARCHY. This crosswalk takes the Legacy Address Type codes and changes them to a numeric value.



This numeric value is then used as an ORDER BY value so that the address type you want to be processed first actually gets processed first. Using the Legacy Address type itself would not work if you wanted MA to be loaded first and then LO as alphabetically that would be the correct order. This hierarchy gives the routine the added benefit if the two records are actually the same exact address but had the Legacy Address Types of LO and MA assigned to it. The function would load the MA Legacy Address Type record first under the Banner Address Type of MA (also doing any appropriate merging with existing Banner MA Address Type records), then attempt to load the LO Legacy Address Type under the Banner Address Type of MA – but here would find an exact match to a pre-existing Banner MA Address Type (the one we just loaded previously) and flag the second record (the LO Address) as an exact match in the temp table SPRADDR_CVT by placing ‘m’ in the SPRADDR_UPD_STATUS custom column

(the function also updates the spraddr_seqno value so that the SPRTELE conversion can fetch the proper link to its address record).

Address Updating by Type and Person States
This function has the further capability of enforcing an override to the decision based on specific Address Types and Person States (merge_string value).We utilize an ATYP_OVERRIDE crosswalk concept which GPSynch can read and determine if for this merge_string of a person for this address type. If a match is found in the crosswalk, then Banner retains system of record regardless of what was retrieved from the grid. This functionality is optional and if not configured would simply be ignored.



If you need to do any special data processing, crosswalking, etc. Then these calls need to be placed in the function code as the functions always read from the raw data columns of the temp table.

It is recommended that you review the source code of the function so that you understand the logic. Currently the functions are designed using IF-THEN-ELSE-END IF which can become confusing.

SPRTELE

F_PRE_LOOP_WRAPTEL.SQL

This function is utilized with the table SPRTELE. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function actually houses calls to three (3) other functions needed to process the SPRTELE table using this methodology. Other custom functions could be added to this function if needed.

This function accepts two (2) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

system_converting – this is the legacy system you are converting. Valid values are: SIS, HRS, FRS, ADS

This function then calls the following three (3) functions:

F_PRE_LOOP_EXT_REFRESH.SQL
F_PRE_LOOP_IND_REFRESH.SQL
F_PRE_LOOP_TELE_MERGE.SQL

These functions will be discussed next.

F_PRE_LOOP_EXT_REFESH.SQL

This function is utilized with the table SPRTELE. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (sprtele_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function refreshes the Oracle table WHO_WINS from the Oracle External Table WHO_WINS_EXT and the Oracle table SUSPENSE_OVERRIDE from the Oracle External Table SUSPENSE_OVERRIDE_EXT. This was done for two reasons: (1) Oracle External tables can not be indexed (2) to reduce I/O on the OS and stop large log files from being produced on the OS.

We want to use the Oracle External table method so that the Grid rules can update dynamically – reducing the need for extra steps to get the merge code to recognize rules changes over time.

It is recommended that you review the source code of the function so that you understand the logic.

F_PRE_LOOP_IND_REFESH.SQL

This function is utilized with the table SPRTELE. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (sprtele_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function ensures the necessary indexes exist for the methodology.

It is recommended that you review the source code of the function so that you understand the logic.

F_PRE_LOOP_TELE_MERGE.SQL

This function is utilized with the table SPRTELE. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (sprtele_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts three (3) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

system_converting – this is the legacy system you are converting. Valid values are: SIS, HRS, FRS, ADS

enrollment_term – This value is optional and not needed when the implementing legacy system is SIS and Admissions is not live. Other systems can provide this value to determine (1) a currently enrolled student by this cut off term and/or (2) a recent applicant in Admissions. ***However, when the implementing legacy system is SIS – the extraction utility of the legacy data must place the word "CURRENT" in the CONVERT_DATA_ORIGIN column of the temp table SPRIDEN_CVT for those ID's that the institution deems to be current students (each institution will have to define what a "current student" actually means). This is the only way we will be able to determine a "current student" as Banner Student is being implemented and is, for all intensive purposes, empty.***

This function only pulls record sets from SPRTELE_CVT that have been flagged as "matched" by setting the sprtele_cvt_status = 'M' in the temp table sprtele_cvt. The function then fetches the Merge String from SPRIDEN_CVT that had already been derived from the merging of Matched SPRIDEN records. This merge string is then used to read the grid to determine if the Banner data takes precedence over the Legacy File data or vice-versa. Since SPRTELE is a repeating table, we have the ability to add rows to this table. Therefore, when reading the grid, the function will only ever look at the values placed at the intersection of the SPRTELE_PIDM row and column containing the derived merge string value. However, when an exact match for a FILE telephone number is found against the Banner telephone number, when reading the grid, the function WILL look at the values placed at the intersection of the derived merge string value and the following columns for SPRTELE: SPRTELE_PHONE_EXT, SPRTELE_UNLIST_IND, SPRTELE_COMMENT, SPRTELE_INTL_ACCESS.

The function then takes appropriate action of inserting new records and updating existing records based off the rule “learned” from the grid. Exactly matching data is flagged by placing the value ‘mBU’ – means “match Banner Updated”… in a custom column called SPRTELE_UPD_STATUS (custom columns will be discussed in the IMPLEMENTATION section, values used in the %_UPD_STATUS columns are documented in the appendix).

If the FILE data is not exactly matching then the FILE telephone record is loaded with the next available sequence number for the PIDM and TELE code. This function will fetch the address sequence number from SPRADDR_CVT to maintain the true link between the address and the telephone number as it is likely that when the linked address record was loaded/updated/exactly matched into/in Banner, a new sequence number was derived/determined.

For example:

> You are implementing SIS. Human Resources and Finance have already gone live in Banner. You record was flagged by Common Matching to be a match to a pre-existing record in Banner. The record coming from SIS is for a Current Student and the record in Banner is for a Former Employee. Your institution has defined the grid such that a Current Student’s SIS data can update Human Resources data for a Former Employee.
>
> The merge_string = ‘NOFECSNO’ and the grid returned ‘FIL’. The function determines that the SIS data is different than the current record in Banner. The function fetches the sequence number for the linked address (if a link exists) and the FILE record is loaded using the next available sequence number for telephones.

If you need to do any special data processing, crosswalking, etc. Then these calls need to be placed in the function code as the functions always read from the raw data columns of the temp table.

It is recommended that you review the source code of the function so that you understand the logic. Currently the functions are designed using IF-THEN-ELSE-END IF which can become confusing.

GOREMAL

F_PRE_LOOP_WRAPEML.SQL

This function is utilized with the table GOREMAL. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function actually houses calls to four (4) other functions needed to process the GOREMAL table using this methodology. Other custom functions could be added to this function if needed.

This function accepts one (1) parameters:

This function accepts two (2) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

system_converting – this is the legacy system you are converting. Valid values are: SIS, HRS, FRS, ADS

This function then calls the following four (4) functions:

F_PRE_LOOP_EXT_REFRESH.SQL
F_PRE_LOOP_IND_REFRESH.SQL
F_CVT_WRAP_GOREMAL_DISP_WEB.SQL
F_PRE_LOOP_EMAL_MERGE.SQL

These functions will be discussed next.

F_PRE_LOOP_EXT_REFESH.SQL

This function is utilized with the table GOREMAL. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (goremal_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function refreshes the Oracle table WHO_WINS from the Oracle External Table WHO_WINS_EXT and the Oracle table SUSPENSE_OVERRIDE from the Oracle External Table SUSPENSE_OVERRIDE_EXT. This was done for two reasons: (1) Oracle External tables can not be indexed (2) to reduce I/O on the OS and stop large log files from being produced on the OS.

We want to use the Oracle External table method so that the Grid rules can update dynamically – reducing the need for extra steps to get the merge code to recognize rules changes over time.

It is recommended that you review the source code of the function so that you understand the logic.

F_PRE_LOOP_IND_REFESH.SQL

This function is utilized with the table GOREMAL. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (goremal_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts one (1) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

This function ensures the necessary indexes exist for the methodology.

It is recommended that you review the source code of the function so that you understand the logic.

F_CVT_WRAP_GOREMAL_DISP_WEB.SQL

This function is utilized with the table GOREMAL. As its name indicates, the function is implemented as a Wrap Up function; meaning that this function will be called after the cursor loop in the conversion program. This function was designed to run when the conversion routine (goremal_convert.sql) disposition is 'C' (process_level = 'C') as a clean up routine after the N to C run. However, we get the same effect by calling it as a Pre Loop function on a C to I run.
.

This function accepts no (0) parameters:

This function determines if the email address should be visible on the web. This is an optional program.

It is recommended that you review the source code of the function so that you understand the logic.

F_PRE_LOOP_EMAL_MERGE.SQL

This function is utilized with the table GOREMAL. As its name indicates, the function is implemented as a Pre-Loop function; meaning that this function will be called before the cursor loop in the conversion program. This function only truly launches when the conversion routine (goremal_convert.sql) disposition is 'I' (process_level = 'I') as we would not want to call this routine both when running our conversion program "N to C" and "C to I" – we only need process this data once.

This function accepts three (3) parameters:

process_level – this is the value you enter for disposition when running your conversion program.

system_converting – this is the legacy system you are converting. Valid values are: SIS, HRS, FRS, ADS

enrollment_term – This value is optional and not needed when the implementing legacy system is SIS and Admissions is not live. Other systems can provide this value to determine (1) a currently enrolled student by this cut off term and/or (2) a recent applicant in Admissions. ***However, when the implementing legacy system is SIS – the extraction utility of the legacy data must place the word "CURRENT" in the CONVERT_DATA_ORIGIN column of the temp table SPRIDEN_CVT for those ID's that the institution deems to be current students (each institution will have to define what a "current student" actually means). This is the only way we will be able to determine a "current student" as Banner Student is being implemented and is, for all intensive purposes, empty.***

This function only pulls record sets from GOREMAL_CVT that have been flagged as "matched" by setting the goremal_cvt_status = 'M' in the temp table goremal_cvt. The function then fetches the Merge String from SPRIDEN_CVT that had already been derived from the merging of Matched SPRIDEN records. This merge string is then used to read the grid to determine if the Banner data takes precedence over the Legacy File data or vice-versa. Since GOREMAL is a repeating table, we have the ability to add rows to this table. Therefore, when reading the grid, the function will only ever look at the values placed at the intersection of the GOREMAL_PIDM row and column containing the derived merge string value. However, when an exact match for a FILE email address is found against the Banner email address, when reading the grid, the function WILL look at the values placed at the intersection of the derived merge string value and the following columns for GOREMAL: GOREMAL_PREFERRED_IND, GOREMAL_STATUS_IND, GOREMAL_COMMENT, GOREMAL_DISP_WEB_IND.

The function then takes appropriate action of inserting new records and updating existing records based off the rule “learned” from the grid. Exactly matching data is flagged by placing the value ‘mBU’ – means “match Banner Updated”… in a custom column called GOREMAL_UPD_STATUS (custom columns will be discussed in the IMPLEMENTATION section, values used in the %_UPD_STATUS columns are documented in the appendix).

For example:

> You are implementing SIS. Human Resources and Finance have already gone live in Banner. You record was flagged by Common Matching to be a match to a pre-existing record in Banner. The record coming from SIS is for a Current Student and the record in Banner is for a Former Employee. Your institution has defined the grid such that a Current Student’s SIS data can update Human Resources data for a Former Employee.
>
> The merge_string = ‘NOFECSNO’ and the grid returned ‘FIL’. The function determines that the SIS data is different than the current record in Banner. The FILE record is loaded.

If you need to do any special data processing, crosswalking, etc. Then these calls need to be placed in the function code as the functions always read from the raw data columns of the temp table.

It is recommended that you review the source code of the function so that you understand the logic. Currently the functions are designed using IF-THEN-ELSE-END IF which can become confusing.

IMPLEMENTATION

In order to implement the **GPSynch** methodology, you will need to have a method for determining that the FILE record is an exact match to one and only one Banner record. The **GPSynch** methodology was designed to work with the F_PRE_LOOP_CM_API.SQL function delivered with the Converter Tool. However, as long as the records in the 5 core temp tables (SPRIDEN_CVT, SPBPERS_CVT, SPRADDR_CVT, SPRTELE_CVT, GOREMAL_CVT) are flagged as 'M' in the <table_name>_cvt_status column and the exactly "matched-to" record information from Banner is loaded into table CURCERR in the same manner the F_PRE_LOOP_CM_API.SQL function would, the process will not know the difference. It is recommended that all 5 core tamp tables are loaded(they must all exist) at time of duplicate checking and the <table_name>_cvt_status is set appropriately for the record set across all tables when a match is found.

Lastly, the assumption is that you are using the Converter Tool for your conversion processing. If not using the Converter Tool, the database objects used would need to be (and could be) created as a standalone for this process.

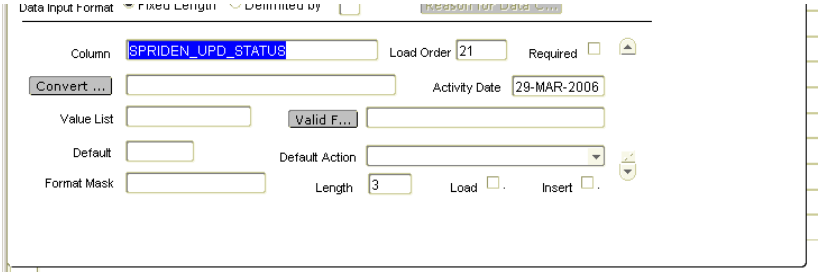
Before insertion into Banner we will be merging exact matches. To accommodate that we will need to add new custom rows to the five core tables we will use for common matching.

NOTE: the installation script gp_synch.sql(sh) for the GPS methodology will do this – use this information as a guide for installation verification.

Add the following to Converter Tool definition for table SPRIDEN:

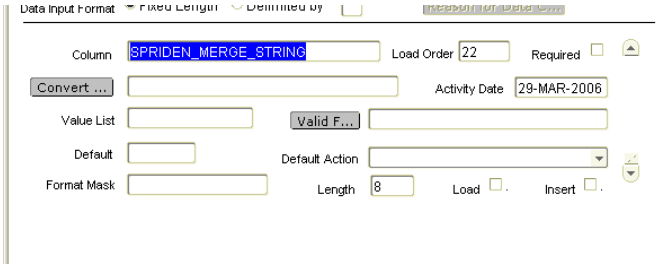
SPRIDEN_UPD_STATUS

load order 21 (or next available)
length = 3
required unchecked
load unchecked
insert unchecked
default action blank



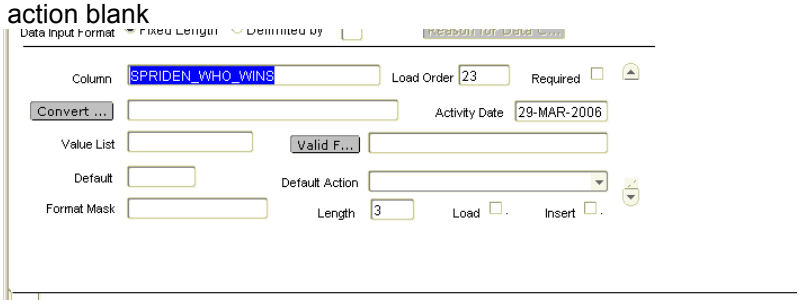
SPRIDEN_MERGE_STRING

load order 22 (or next available)
length = 8
required unchecked
load unchecked
insert unchecked
default action blank



SPRIDEN_WHO_WINS

load order 23 (or next available)
length = 3
required unchecked
load unchecked
insert unchecked
default action blank



Next, you will want to add the Pre Loop function to the Wrapup Function field in the Converter Tool. Generally, there are several pre-loop and wrap-up function calls needed for SPRIDEN, so a larger pre-loop function will need to be utilized to accommodate the multiple calls.

Below is the recommended modification to F_PREWP.SQL:

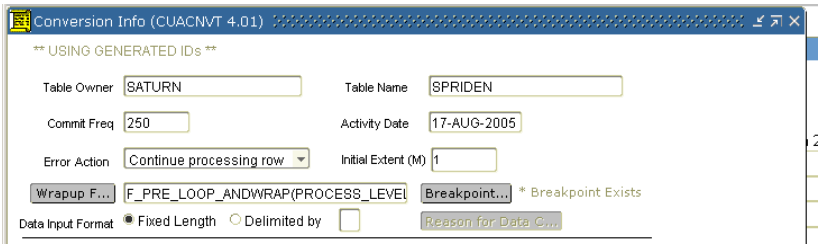
```
CREATE OR REPLACE FUNCTION F_PREWP
        (process_level varchar2, cmsc_code_in varchar2,
               records_in varchar2,cur_jobid NUMBER)

  return varchar2
IS
--
-- FILE NAME..: F_PRE_LOOP_WRP
-- RELEASE....:
-- OBJECT NAME: F_PRE_LOOP_WRP
-- PRODUCT....: SCTCVT
-- USAGE......:
-- COPYRIGHT..:
--
-- DESCRIPTION:
--    Pre Loop function that calls functions to accomodate both a PRE and WRAP
--
-- pass in process_level variable then the Common Matching rule code
-- then cur_jobid variable call should look like this:
--   f_prewp(process_level,'CONV',records_in,cur_jobid)
-- watch out as call can't be longer than 50 characters
--
-- DESCRIPTION END
--
-- AUDIT TRAIL:
-- 20061024 - new version of functions delivered
--
--
   ws_insert_count       number;
   ws_error_count        number;
   ws_dummy              varchar2(4000);
BEGIN
 -- This is to refresh CURCERR table.
 BEGIN
   IF records_in = 'N' AND process_level = 'C' THEN
        ws_dummy := F_CVT_CURCERR_CLEAN('Y'); -- will truncate curcerr
        --ws_dummy := F_CVT_CURCERR_CLEAN('N'); -- will NOT truncate curcerr
        IF ws_dummy like 'ERR-%' or ws_dummy like 'ORA-%'  THEN
         return 'ERR-'||ws_dummy;
        END IF;
   END IF;
 END;
 -- This is to refresh WHO_WINS and  SUSPENSE_OVERRIDE table.
 BEGIN
   IF process_level = 'C' THEN
        ws_dummy := F_PRE_LOOP_EXT_REFRESH('I');
        IF ws_dummy like 'ERR-%' or ws_dummy like 'ORA-%'  THEN
         return 'ERR-'||ws_dummy;
        END IF;
   END IF;
 END;
 -- This is to ensure proper indexes exist.
 BEGIN
   IF process_level = 'C' THEN
        ws_dummy := F_PRE_LOOP_IND_REFRESH('I');
        IF ws_dummy like 'ERR-%' or ws_dummy like 'ORA-%'  THEN
         return 'ERR-'||ws_dummy;
        END IF;
   END IF;
 END;
 -- This is to check for duplicates in Banner using Common matching
```

```
 BEGIN
   IF records_in = 'N' and process_level = 'C' THEN
        ws_dummy :=
F_PRE_LOOP_CM_API(process_level,cmsc_code_in,records_in,cur_jobid);
       IF ws_dummy like 'ERR-%' or ws_dummy like 'ORA-%' THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
 -- This is to overide the suspense records depending on value retrieved from SUS-
PENSE_OVERRIDE table.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_SUSPENSE_OVERRIDE(process_level);
       IF ws_dummy like 'ERR-%' or ws_dummy like 'ORA-%'  THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
 -- This is to merge the File records with existing Banner records depending on
value retrieved from WHO_WINS Matrix.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_IDEN_MERGE(process_level,'SIS');
       IF ws_dummy like 'ERR-%' or ws_dummy like 'ORA-%'  THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
 -- This is to fix the Name change records for non-duplicates so that the API will
load them under the new Banner Generated ID.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_WRAP_CURR_ID_FOR_NAME_CHG('C');
       IF ws_dummy like 'ERR-%' or ws_dummy like 'ORA-%'  THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
 -- This is to fix the ID change records for non-duplicates so that the API will
load them.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_WRAP_CURR_NAME_FOR_ID_CHG('C','F');
       IF ws_dummy like 'ERR-%' or ws_dummy like 'ORA-%'  THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
  commit;

  return 'Successful completion of F_PREWP';

EXCEPTION
  WHEN OTHERS THEN
     RETURN SUBSTR('ERR- in F_PREWP '||SQLERRM, 1,200);

END F_PREWP;
/
SHOW ERRORS
```

**note that ‘SIS’ should be replaced with the Legacy System you are implementing. Valid values are FRS, HRS, SIS, ADS.

**note that ‘CONV’ should be replaced with the Common Matching rule you are using – see function example source code

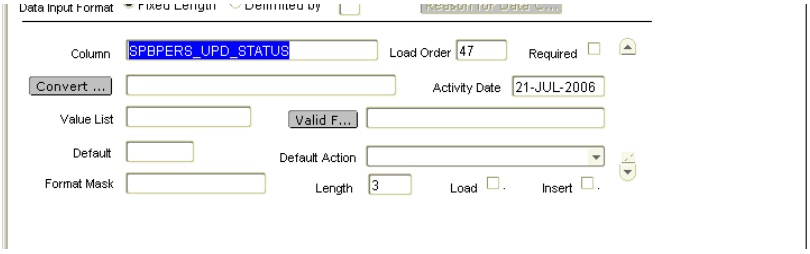
This function call **F_PREWP(PROCESS_LEVEL,'SIS',RECORDS_IN,CUR_JOBID)** should be placed in the Converter Tool rules (see screenshot):



Add the following to Converter Tool definition for table SPBPERS:

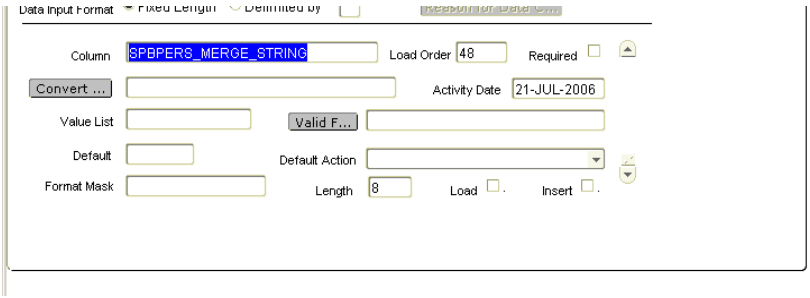
SPBPERS_UPD_STATUS

load order 47 (or next available)
length = 3
required unchecked
load unchecked
insert unchecked
default action blank



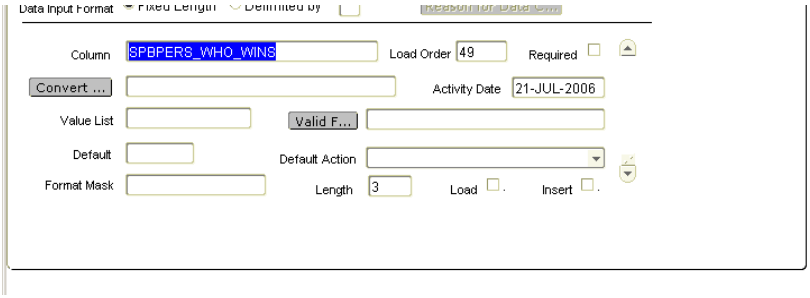
SPBPERS_MERGE_STRING

load order 48 (or next available)
length = 8
required unchecked
load unchecked
insert unchecked
default action blank



SPBPERS_WHO_WINS

load order 49 (or next available)
length = 3
required unchecked
load unchecked
insert unchecked
default action blank



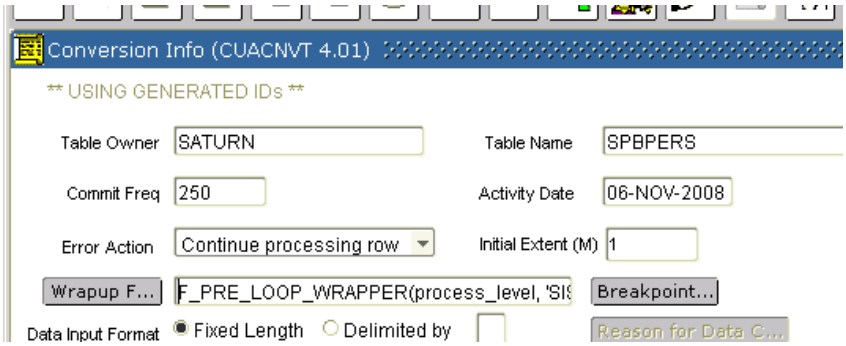
Next, you will want to add the Pre Loop function to the Wrapup Function field in the Converter Tool. If there are several pre-loop and wrap-up function calls needed for SPBPERS, a larger pre-loop function will need to be utilized to accommodate the multiple calls as was done for SPRIDEN. Below is a SAMPLE wrapper function.

Below is the recommended modification to F_PRE_LOOP_WRAPPER.SQL:

```
CREATE OR REPLACE FUNCTION F_PRE_LOOP_WRAPPER
             (process_level varchar2,
         system_converting IN varchar2)

  return varchar2
IS
--
-- FILE NAME..: F_PRE_LOOP_WRAPPER
-- RELEASE....:
-- OBJECT NAME: F_PRE_LOOP_WRAPPER
-- PRODUCT....: SCTCVT
-- USAGE......:
-- COPYRIGHT..:
--
-- DESCRIPTION:
--    Pre Loop function that calls functions to accomodate both a PRE and WRAP
--
--
-- DESCRIPTION END
--
-- AUDIT TRAIL:
--
--
--
   ws_insert_count       number;
   ws_error_count        number;
   ws_dummy              varchar2(4000);
BEGIN
 -- This is to refresh WHO_WINS and  SUSPENSE_OVERRIDE table.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_EXT_REFRESH(process_level);
       IF ws_dummy like 'ERR-%' THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
 -- This is to ensure proper indexes exist.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_IND_REFRESH(process_level);
       IF ws_dummy like 'ERR-%' THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
 -- This is to merge the File records with existing Banner records depending on
value retrieved from WHO_WINS Matrix.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_TELE_MERGE(process_level,'SIS');
       IF ws_dummy like 'ERR-%' THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;

  commit;
```

```
  return 'Successful completion of F_PRE_LOOP_WRAPPER';

EXCEPTION
  WHEN OTHERS THEN
     RETURN SUBSTR('ERR- in F_PRE_LOOP_WRAPPER '||SQLERRM, 1,200);

END F_PRE_LOOP_WRAPPER;
/
SHOW ERRORS
```

This function call **F_PRE_LOOP_WRAPPER(process_level,'SIS')** should be placed in the Converter Tool rules (see screenshot):

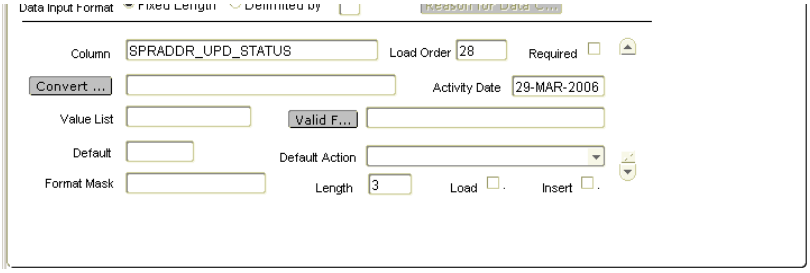


**note that 'SIS' should be replaced with the Legacy System you are implementing. Valid values are FRS, HRS, SIS, ADS

Add the following to Converter Tool definition for table SPRADDR:

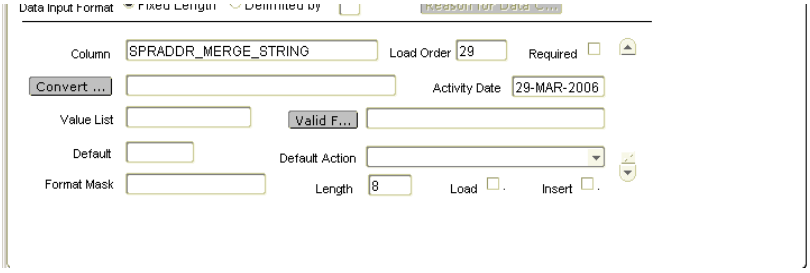
SPRADDR_UPD_STATUS

load order 28 (or next available)
length = 3
required unchecked
load unchecked
insert unchecked
default action blank



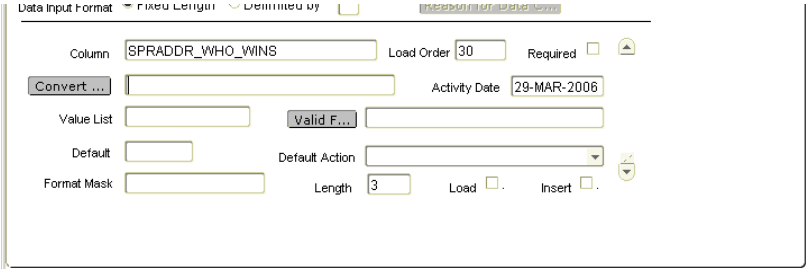
SPRADDR_MERGE_STRING

load order 29 (or next available)
length = 8
required unchecked
load unchecked
insert unchecked
default action blank



SPRADDR_WHO_WINS

load order 30 (or next available)
length = 3
required unchecked
load unchecked
insert unchecked
default action blank

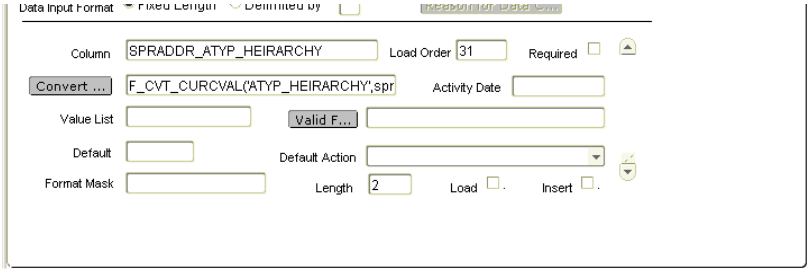


SPRADDR_ATYP_HEIRARCHY

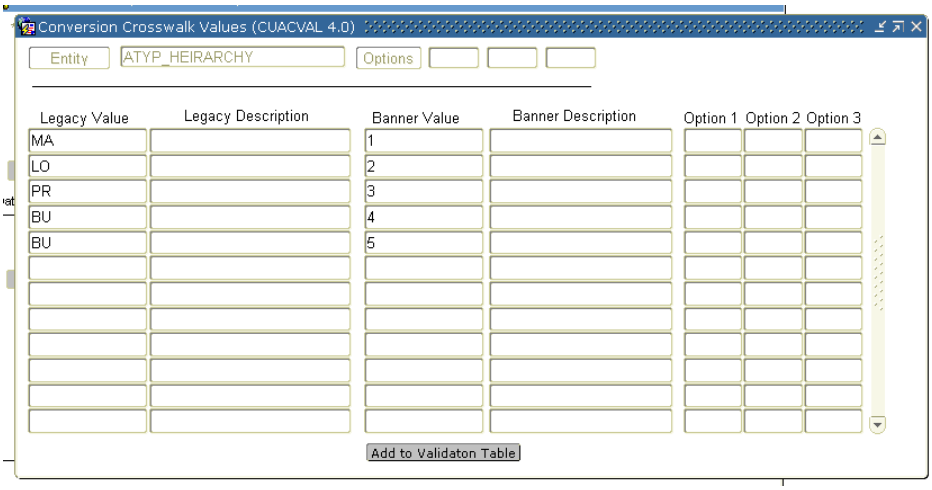
load order 31 (or next available)
length = 2
required unchecked
load unchecked
insert unchecked
default action blank
convert function -

F_CVT_CURCVAL('ATYP_HEIRARCHY',spraddr_rec.convert_atyp_code)

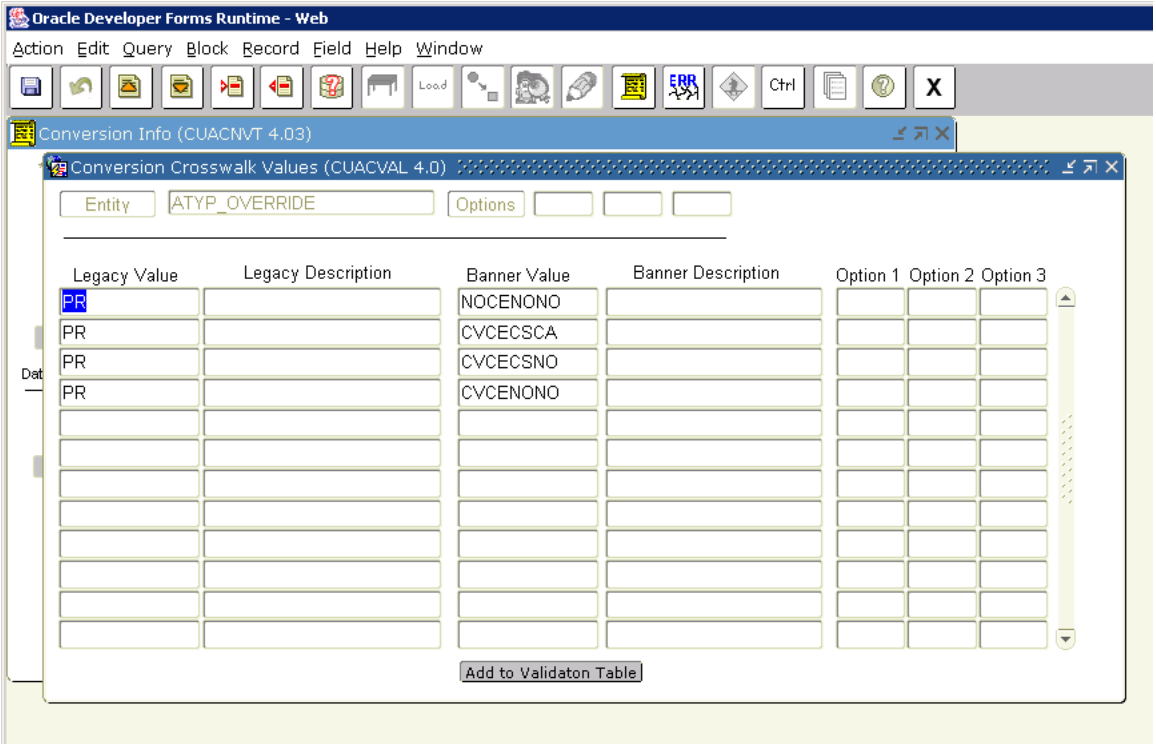
** this is what will determine the hierarchy of the address types for insertion into banner.



You may need to create a crosswalk called ATYP_HEIRARCHY with the legacy Address Type crosswalked to a number value that dictates the precedence, but you only need to do this if you are combining legacy Address Type codes to one in Banner.



You may need to build the ATYP_OVERRIDE crosswalk if you need to use the “Address Updating by Type and Person State” functionality.



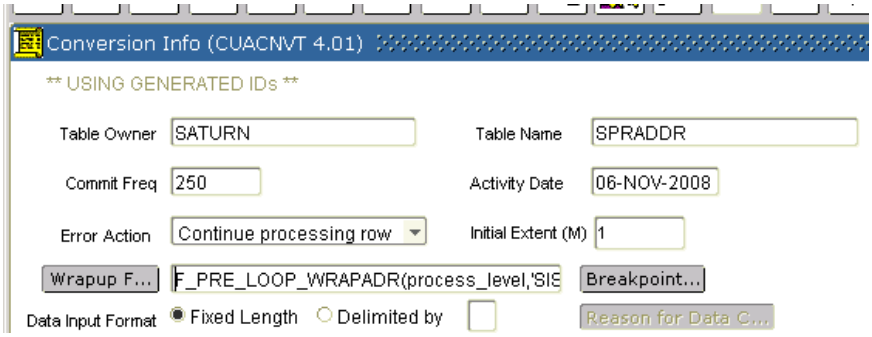
Next, you will want to add the Pre Loop function to the Wrapup Function field in the Converter Tool. If there are several pre-loop and wrap-up function calls needed for SPRADDR, a larger pre-loop function will need to be utilized to accommodate the multiple calls as was done for SPRIDEN. Below is a SAMPLE wrapper function.

Below is the recommended modification to F_PRE_LOOP_WRAPADR.SQL:

```
CREATE OR REPLACE FUNCTION F_PRE_LOOP_WRAPADR
        (process_level varchar2,
         system_converting IN varchar2)

  return varchar2
IS
--
-- FILE NAME..: F_PRE_LOOP_WRAPADR
-- RELEASE....:
-- OBJECT NAME: F_PRE_LOOP_WRAPADR
-- PRODUCT....: SCTCVT
-- USAGE......:
-- COPYRIGHT..:
--
-- DESCRIPTION:
--    Pre Loop function that calls functions to accomodate both a PRE and WRAP
--
--
-- DESCRIPTION END
--
-- AUDIT TRAIL:
--
--
--
   ws_insert_count       number;
   ws_error_count        number;
```

```
   ws_dummy              varchar2(4000);
BEGIN
 -- This is to refresh WHO_WINS and  SUSPENSE_OVERRIDE table.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_EXT_REFRESH(process_level);
        IF ws_dummy like 'ERR-%' THEN
         return 'ERR-'||ws_dummy;
        END IF;
   END IF;
 END;
 -- This is to ensure proper indexes exist.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_IND_REFRESH(process_level);
        IF ws_dummy like 'ERR-%' THEN
         return 'ERR-'||ws_dummy;
        END IF;
   END IF;
 END;
 -- This is to merge the File records with existing Banner records depending on
value retrieved from WHO_WINS Matrix.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_ADDR_MERGE(process_level,'SIS');
        IF ws_dummy like 'ERR-%' THEN
         return 'ERR-'||ws_dummy;
        END IF;
   END IF;
 END;
  commit;

  return 'Successful completion of F_PRE_LOOP_WRAPADR';

EXCEPTION
  WHEN OTHERS THEN
     RETURN SUBSTR('ERR- in F_PRE_LOOP_WRAPADR '||SQLERRM, 1,200);

END F_PRE_LOOP_WRAPADR;
/
SHOW ERRORS
```

This function call **F_PRE_LOOP_WRAPADR(process_level,'SIS')** should be placed in the Converter Tool rules (see screenshot):

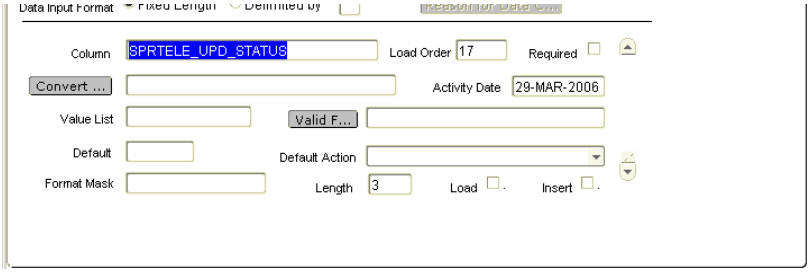


**note that 'SIS' should be replaced with the Legacy System you are implementing. Valid values are FRS, HRS, SIS, ADS

Add the following to Converter Tool definition for table SPRTELE:

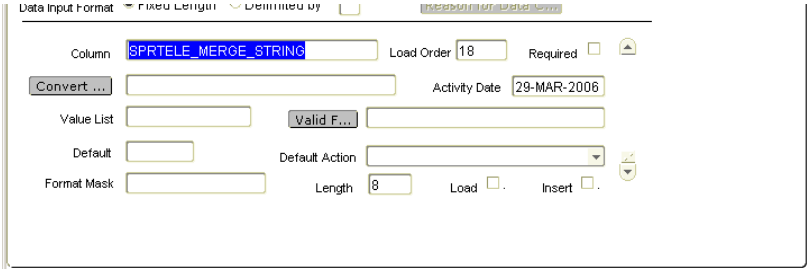
SPRTELE_UPD_STATUS

load order 17 (or next available)
length = 3
required unchecked
load unchecked
insert unchecked
default action blank



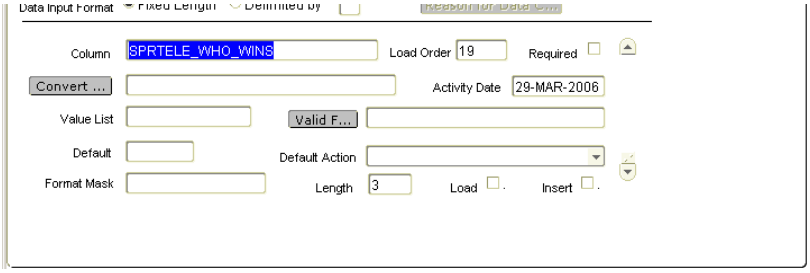
SPRTELE_MERGE_STRING

load order 18 (or next available)
length = 8
required unchecked
load unchecked
insert unchecked
default action blank



SPRTELE_WHO_WINS

load order 19 (or next available)
length = 3
required unchecked
load unchecked
insert unchecked
default action blank



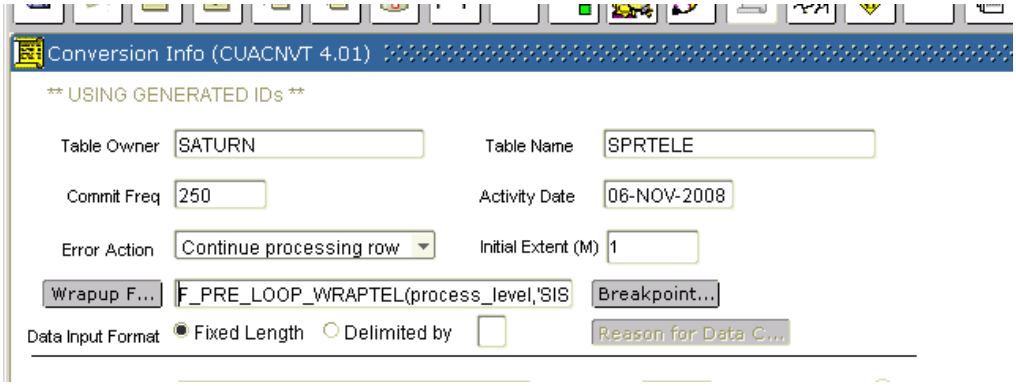
Next, you will want to add the Pre Loop function to the Wrapup Function field in the Converter Tool. If there are several pre-loop and wrap-up function calls needed for SPRTELE, a larger pre-loop function will need to be utilized to accommodate the multiple calls as was done for SPRIDEN. Below is a SAMPLE wrapper function.

Below is the recommended modification to F_PRE_LOOP_WRAPTEL.SQL:

```
CREATE OR REPLACE FUNCTION F_PRE_LOOP_WRAPTEL
        (process_level varchar2,
         system_converting IN varchar2)
  return varchar2
IS
--
-- FILE NAME..: F_PRE_LOOP_WRAPTEL
-- RELEASE....:
-- OBJECT NAME: F_PRE_LOOP_WRAPTEL
-- PRODUCT....: SCTCVT
-- USAGE......:
-- COPYRIGHT..:
--
-- DESCRIPTION:
--    Pre Loop function that calls functions to accomodate both a PRE and WRAP
--
--
-- DESCRIPTION END
--
-- AUDIT TRAIL:
--
--
--
   ws_insert_count       number;
   ws_error_count        number;
   ws_dummy              varchar2(4000);
BEGIN
 -- This is to refresh WHO_WINS and  SUSPENSE_OVERRIDE table.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_EXT_REFRESH(process_level);
       IF ws_dummy like 'ERR-%' THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
 -- This is to ensure proper indexes exist.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_IND_REFRESH(process_level);
       IF ws_dummy like 'ERR-%' THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
 -- This is to merge the File records with existing Banner records depending on
value retrieved from WHO_WINS Matrix.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_TELE_MERGE(process_level,'SIS');
       IF ws_dummy like 'ERR-%' THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
  commit;

  return 'Successful completion of F_PRE_LOOP_WRAPTEL';
```

```
EXCEPTION
  WHEN OTHERS THEN
     RETURN SUBSTR('ERR- in F_PRE_LOOP_WRAPTEL '||SQLERRM, 1,200);

END F_PRE_LOOP_WRAPTEL;
/
SHOW ERRORS
```

This function **F_PRE_LOOP_WRAPTEL(process_level,'SIS')** call should be placed in the Converter Tool rules (see screenshot):

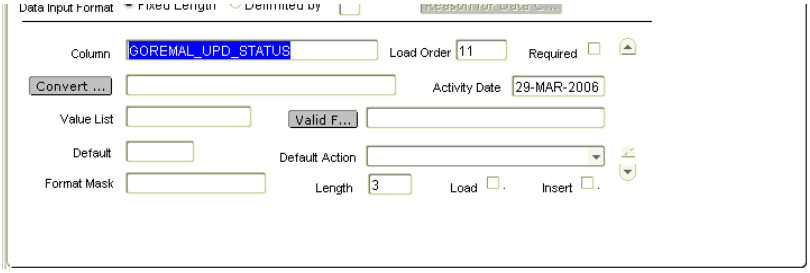


**note that 'SIS' should be replaced with the Legacy System you are implementing. Valid values are FRS, HRS, SIS, ADS

Add the following to Converter Tool definition for table GOREMAL:

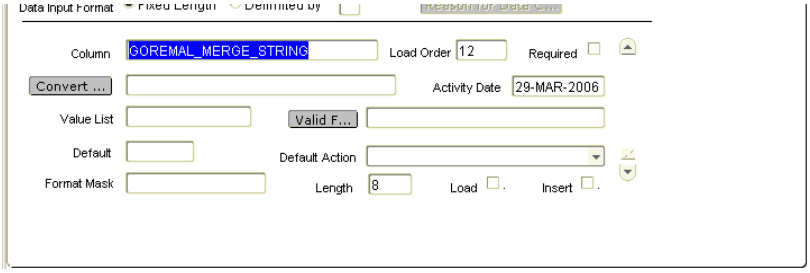
GOREMAL_UPD_STATUS

load order 11 (or next available)
length = 3
required unchecked
load unchecked
insert unchecked
default action blank



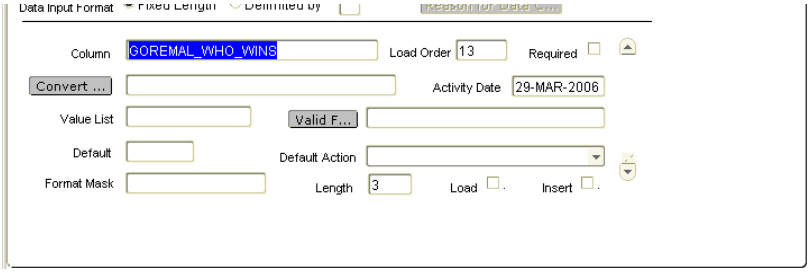
GOREMAL_MERGE_STRING

load order 12 (or next available)
length = 8
required unchecked
load unchecked
insert unchecked
default action blank



GOREMAL_WHO_WINS

load order 13 (or next available)
length = 3
required unchecked
load unchecked
insert unchecked
default action blank



Next, you will want to add the Pre Loop function to the Wrapup Function field in the Converter Tool. If there are several pre-loop and wrap-up function calls needed for SPRTELE, a larger pre-loop function will need to be utilized to accommodate the multiple calls as was done for SPRIDEN. Below is a SAMPLE wrapper function.

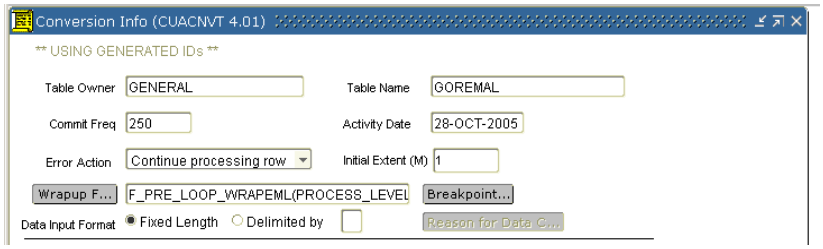
Below is the recommended modification to F_PRE_LOOP_WRAPEML.SQL:

```
CREATE OR REPLACE FUNCTION F_PRE_LOOP_WRAPEML
        (process_level varchar2,
         system_converting IN varchar2)
  return varchar2
IS
--
-- FILE NAME..: F_PRE_LOOP_WRAPEML
-- RELEASE....:
-- OBJECT NAME: F_PRE_LOOP_WRAPEML
-- PRODUCT....: SCTCVT
-- USAGE......:
-- COPYRIGHT..:
--
-- DESCRIPTION:
--    Pre Loop function that calls functions to accomodate both a PRE and WRAP
--
--
-- DESCRIPTION END
--
-- AUDIT TRAIL:
--
--
--
   ws_insert_count       number;
   ws_error_count        number;
   ws_dummy              varchar2(4000);
BEGIN
 -- This is to refresh WHO_WINS and  SUSPENSE_OVERRIDE table.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_EXT_REFRESH(process_level);
       IF ws_dummy like 'ERR-%' THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
 -- This is to ensure proper indexes exist.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_IND_REFRESH(process_level);
       IF ws_dummy like 'ERR-%' THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
 -- This is to first check to see if web display can be enabled
 BEGIN
   IF process_level = 'C' THEN
        ws_dummy := F_CVT_WRAP_GOREMAL_DISP_WEB();
       IF ws_dummy like 'ERR-%' THEN
        return 'ERR-'||ws_dummy;
       END IF;
   END IF;
 END;
 -- This is to merge the File records with existing Banner records depending on
value retrieved from WHO_WINS Matrix.
 BEGIN
   IF process_level = 'I' THEN
        ws_dummy := F_PRE_LOOP_EMAL_MERGE(process_level,'SIS');
```

```
        IF ws_dummy like 'ERR-%' THEN
         return 'ERR-'||ws_dummy;
        END IF;
    END IF;
  END;

   commit;

   return 'Successful completion of F_PRE_LOOP_WRAPEML';

EXCEPTION
   WHEN OTHERS THEN
      RETURN SUBSTR('ERR- in F_PRE_LOOP_WRAPEML '||SQLERRM, 1,200);

END F_PRE_LOOP_WRAPEML;
/
SHOW ERRORS
```

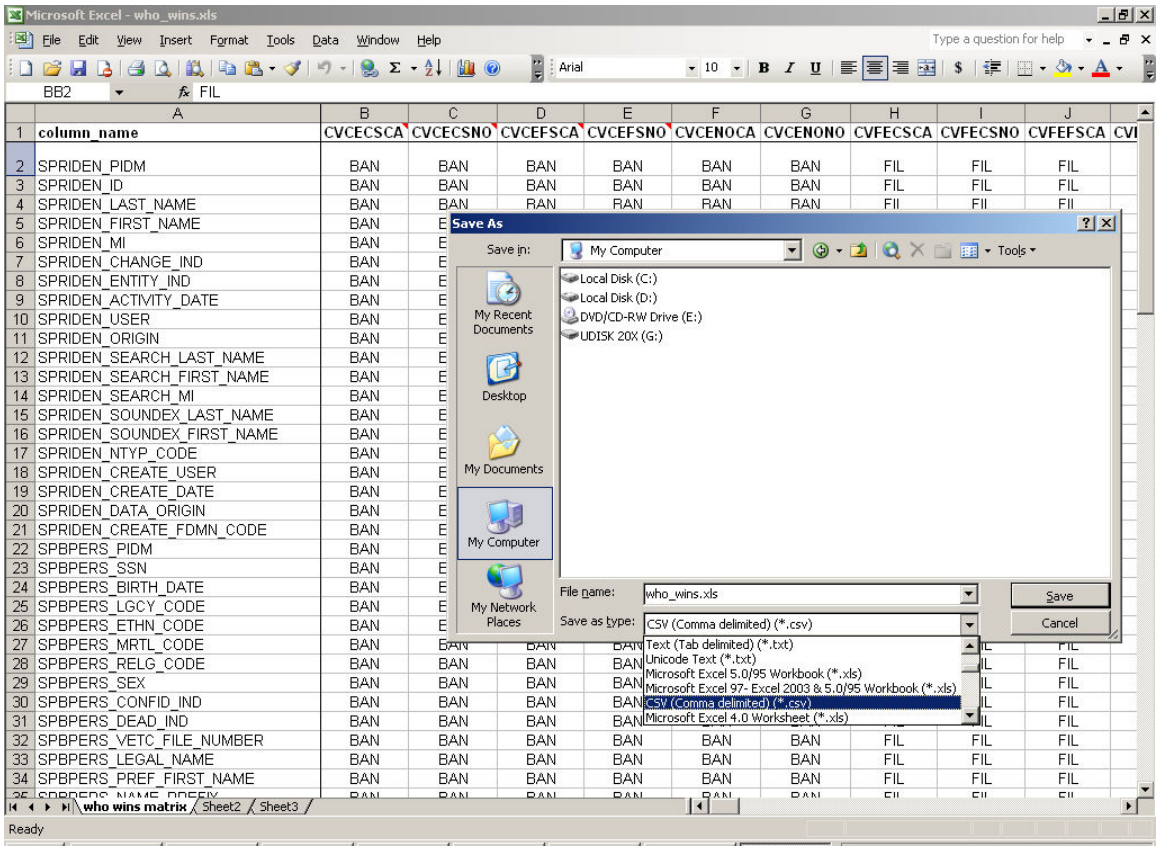
**note that 'SIS' should be replaced with the Legacy System you are implementing. Valid values are FRS, HRS, SIS, ADS

This function call **F_PRE_LOOP_WRAPEML(PROCESS_LEVEL,'SIS')** should be placed in the Converter Tool rules (see screenshot):



Setting up the WHO_WINS grid:

At this point all the Converter Tool Rules have been defined. Next we need to set up the Grid. You should work with decision makers in completing the Grid (who_wins.xls). Once the Grid is completed, you will need to save the Excel spreadsheet as a comma delimited file (choose .csv choice when performing a SAVE AS):



**note: when creating the who_wins.csv from the who_wins.xls - remove the comments columns from the .csv file

Next, place the “who_wins.csv file in the appropriate OS directory. For example:

/shared/appldev/common/ctool/stucvt/genpers

Next you will create the Oracle Directory in SQL so that your Oracle External table will know where to find the Grid (or who_wins.csv) and Suspense Override information.

Syntax:

Create or replace directory ext_merge_table as ‘<os_path_name>’;

From our example OS directory above we would type:

```
create or replace directory ext_merge_table as '/shared/appldev/common/ctool/stucvt/genpers';
```

Next, we create the Oracle External Tables by running the following in SQL:

```
create table who_wins_ext
(column_name varchar2(60),
CVCECSCA        varchar2(3),
CVCECSNO        varchar2(3),
CVCEFSCA        varchar2(3),
CVCEFSNO        varchar2(3),
CVCENOCA        varchar2(3),
CVCENONO        varchar2(3),
CVFECSCA        varchar2(3),
CVFECSNO        varchar2(3),
CVFEFSCA        varchar2(3),
CVFEFSNO        varchar2(3),
CVFENOCA        varchar2(3),
CVFENONO        varchar2(3),
CVNOCSCA        varchar2(3),
CVNOCSNO        varchar2(3),
CVNOFSCA        varchar2(3),
CVNOFSNO        varchar2(3),
CVNONOCA        varchar2(3),
CVNONONO        varchar2(3),
FVCECSCA        varchar2(3),
FVCECSNO        varchar2(3),
FVCEFSCA        varchar2(3),
FVCEFSNO        varchar2(3),
FVCENOCA        varchar2(3),
FVCENONO        varchar2(3),
FVFECSCA        varchar2(3),
FVFECSNO        varchar2(3),
FVFEFSCA        varchar2(3),
FVFEFSNO        varchar2(3),
FVFENOCA        varchar2(3),
FVFENONO        varchar2(3),
FVNOCSCA        varchar2(3),
FVNOCSNO        varchar2(3),
FVNOFSCA        varchar2(3),
FVNOFSNO        varchar2(3),
FVNONOCA        varchar2(3),
FVNONONO        varchar2(3),
NOCECSCA        varchar2(3),
NOCECSNO        varchar2(3),
NOCEFSCA        varchar2(3),
NOCEFSNO        varchar2(3),
NOCENOCA        varchar2(3),
NOCENONO        varchar2(3),
NOFECSCA        varchar2(3),
NOFECSNO        varchar2(3),
NOFEFSCA        varchar2(3),
NOFEFSNO        varchar2(3),
```

```
NOFENOCA        varchar2(3),
NOFENONO        varchar2(3),
NONOCSCA        varchar2(3),
NONOCSNO        varchar2(3),
NONOFSCA        varchar2(3),
NONOFSNO        varchar2(3),
NONONOCA        varchar2(3),
NONONONO        varchar2(3))
  ORGANIZATION EXTERNAL
   (TYPE ORACLE_LOADER
    DEFAULT DIRECTORY ext_merge_table
    ACCESS PARAMETERS
      (RECORDS DELIMITED BY NEWLINE
       FIELDS TERMINATED BY ','
       MISSING FIELD VALUES ARE NULL)
    LOCATION ('who_wins.csv')
    )
  REJECT LIMIT UNLIMITED;

create table suspense_override_ext
(error_message_to_override varchar2(1000),
new_cvt_status  varchar2(1))
  ORGANIZATION EXTERNAL
   (TYPE ORACLE_LOADER
    DEFAULT DIRECTORY ext_merge_table
    ACCESS PARAMETERS
      (RECORDS DELIMITED BY NEWLINE
       FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
       MISSING FIELD VALUES ARE NULL)
    LOCATION ('suspense_override.csv')
    )
                                        REJECT LIMIT UNLIMITED;
```

Next, Create temp tables

```
@spriden_cvt_create.sql
@spbpers_cvt_create.sql
@spraddr_cvt_create.sql
@sprtele_cvt_create.sql
@goremal_cvt_create.sql
```

Next, compile functions for merging:

```
@f_cvt_flag_match.sql
@f_pre_loop_cm_api.sql
@f_pre_loop_iden_merge.sql
@f_pre_loop_pers_merge.sql
@f_pre_loop_addr_merge.sql
@f_pre_loop_tele_merge.sql
@f_pre_loop_emal_merge.sql
@f_suspense_override.sql
@f_pre_loop_ext_refresh.sql
```

```
@f_pre_loop_ind_refresh.sql
@f_wrap_curr_name_for_id_chg.sql
@f_cvt_curcerr_clean.sql
@f_pre_loop_wrp.sql
@f_pre_loop_wrapper.sql
@f_pre_loop_wrapadr.sql
@f_pre_loop_wraptel.sql
@f_pre_loop_wrapeml.sql
```

The following indexes will improve performance of the merging functions:

```
create index temp_index1 on saturn.spriden(spriden_id);
create index dupe_index1 on spriden_cvt(convert_id);
create index dupe_index2 on spriden_cvt(convert_pidm);
create index dupe_index3 on spbpers_cvt(convert_pidm, spbpers_cvt_status);
create index dupe_index4 on spraddr_cvt(convert_pidm, spraddr_cvt_status);
create index dupe_index5 on sprtele_cvt(convert_pidm, sprtele_cvt_status);
create index dupe_index6 on goremal_cvt(convert_pidm, goremal_cvt_status);
create index dupe_index7 on spriden_cvt(convert_pidm, spriden_cvt_status);
create index dupe_index8 on spriden_cvt(convert_id, spriden_cvt_status);
create index dupe_index9 on spriden_cvt(convert_change_ind, spriden_cvt_status);
create index dupe_index10 on spbpers_cvt(convert_pidm);
create index dupe_index11 on spraddr_cvt(convert_pidm);
create index dupe_index12 on sprtele_cvt(convert_pidm);
create index dupe_index13 on goremal_cvt(convert_pidm);
create index dupe_index14 on curcerr(curcerr_table_owner, curcerr_column_name,
curcerr_legacy_value, curcerr_record_id);
create index dupe_index15 on curcerr(curcerr_table_owner, curcerr_record_id);
create index dupe_index16 on spriden_cvt(convert_id, convert_pidm);
create index dupe_index17 on spriden_cvt(spriden_cvt_record_id, spriden_cvt_status);
--
create index merge_index1 on curcerr(curcerr_record_id, curcerr_table_owner);
create index merge_index2 on saturn.spriden(spriden_pidm, spriden_create_date);
create index merge_index3 on spriden_cvt(convert_pidm, spri-
den_upd_status,spriden_cvt_status, convert_change_ind DESC);
create index merge_index4 on spriden_cvt(convert_id, spri-
den_upd_status,spriden_cvt_status, convert_change_ind DESC);
create index merge_index5 on spbpers_cvt(convert_pidm,
spbpers_upd_status,spbpers_cvt_status);
create index merge_index6 on spraddr_cvt(convert_pidm, spraddr_upd_status,
spraddr_pidm, convert_atyp_code, spraddr_cvt_record_id);
create index merge_index7 on sprtele_cvt(convert_pidm, sprtele_upd_status,
sprtele_pidm, convert_tele_code, convert_atyp_code, sprtele_cvt_record_id);
create index merge_index8 on goremal_cvt(convert_pidm, goremal_upd_status, gore-
mal_pidm, convert_emal_code, goremal_cvt_record_id);
create index merge_index9 on curcerr(curcerr_record_id, curcerr_table_owner,
curcerr_legacy_value);
--
create index iden_merge1 on spriden_cvt(spriden_cvt_status);
--create index iden_merge2 on saturn.spriden(spriden_id, spriden_search_last_name,
spriden_search_first_name, spriden_search_mi, spriden_pidm);
--create index iden_merge3 on saturn.spriden(spriden_id, spriden_pidm);
create index iden_merge4 on saturn.spriden(spriden_search_last_name, spri-
den_search_first_name, spriden_search_mi, spriden_pidm);
--
```

```
create index pers_merge1 on spbpers_cvt(spbpers_cvt_status);
--
create index addr_merge1 on spraddr_cvt(convert_pidm, spraddr_upd_status);
create index addr_merge2 on saturn.spraddr(spraddr_pidm, spraddr_atyp_code);
create index addr_merge3 on saturn.spraddr(spraddr_pidm, spraddr_atyp_code,
spraddr_status_ind);
create index addr_merge4 on spraddr_cvt(spraddr_cvt_status);
create index addr_merge5 on spraddr(spraddr_pidm, spraddr_atyp_code,
spraddr_street_line1, spraddr_street_line2, spraddr_street_line3, spraddr_city,
spraddr_zip, spraddr_status_ind);
create index addr_merge6 on saturn.spraddr(spraddr_pidm, spraddr_atyp_code,
spraddr_seqno);
create index addr_merge7 on saturn.spraddr(spraddr_pidm, spraddr_atyp_code,
spraddr_from_date, spraddr_to_date);
create index addr_merge8 on spraddr_cvt(spraddr_pidm, spraddr_atyp_code);
--
create index tele_merge1 on spraddr_cvt(spraddr_pidm, convert_atyp_code, con-
vert_seqno);
create index tele_merge2 on saturn.sprtele(sprtele_pidm, sprtele_tele_code,
sprtele_phone_number, sprtele_atyp_code, sprtele_addr_seqno);
create index tele_merge3 on sprtele_cvt(convert_pidm, sprtele_upd_status);
create index tele_merge4 on sprtele_cvt(sprtele_cvt_status);
create index tele_merge5 on saturn.sprtele(sprtele_pidm, sprtele_tele_code,
sprtele_phone_number);
--
create index emal_merge1 on goremal_cvt(goremal_cvt_status);
create index emal_merge2 on general.goremal(goremal_pidm, goremal_emal_code,
goremal_email_address);
create index emal_merge3 on general.goremal(goremal_pidm, goremal_preferred_ind);
```

**note: when creating indexes on real BANNER tables – make sure to remove them once this process is complete or no longer used/needed.

You are now ready to begin using **GPSynch**

Keep in mind that you should be reviewing the <table_name>_UPD_STATUS flags and if ERR is returned – then you should report on the error message stored in CURCERR for the merge routine that produced the error. Most times the errors are due to data problems (i.e. overlength address records, missing crosswalks, etc.) and can be easily resolved. However, while these functions have been tested and used in recent Banner implementations, there is always the chance for a logic problem that would require the rework/defect correction of one of these functions – and we should never be so complacent in a process that is so complex.

APPENDIX

BIG BANG Concept

Some Institutions decide that performing a "Big Bang" General Person conversion will work best for them. Following this method; an Institution will convert ALL General Person from ALL Legacy Systems at the beginning of the UDC implementation and then keep the changes from each legacy system synchronized through GPSynch. The Legacy to Banner synchronization usually takes one of two forms:

(1) Running GPSynch at some interval (nightly, weekly, monthly), then retiring each systems GPSynch as each Banner system becomes fully LIVE and legacy systems are shut down or become view only
(2) Perform a mass synchronization for that Legacy system to Banner in a "just – in –time" fashion, just prior to that systems go-live.

This method/concept raises some issues on how we determine the Status of an individual in Banner. Since only General Person data was loaded to Banner… GPSynch will not have the ability to determine if an individual is a Vendor , an Employee, a Student, an Alum.

In order to bridge this gap until systems are live and GPSynch can then determine an individual's status in the other systems; We will use Business Rules and Roles.

First, we must define in Banner a Business Rule Process Code of “GPSYNCH”, a description of “GPSynch Big Bang Enhancement”, leave the SYSTEM REQUIRED unchecked, and default today’s date for “Start Date” (see screen shot below):

Oracle Developer Forms Runtime - Web: Open > GTVSQPR

File Edit Options Block Item Record Query Tools Help

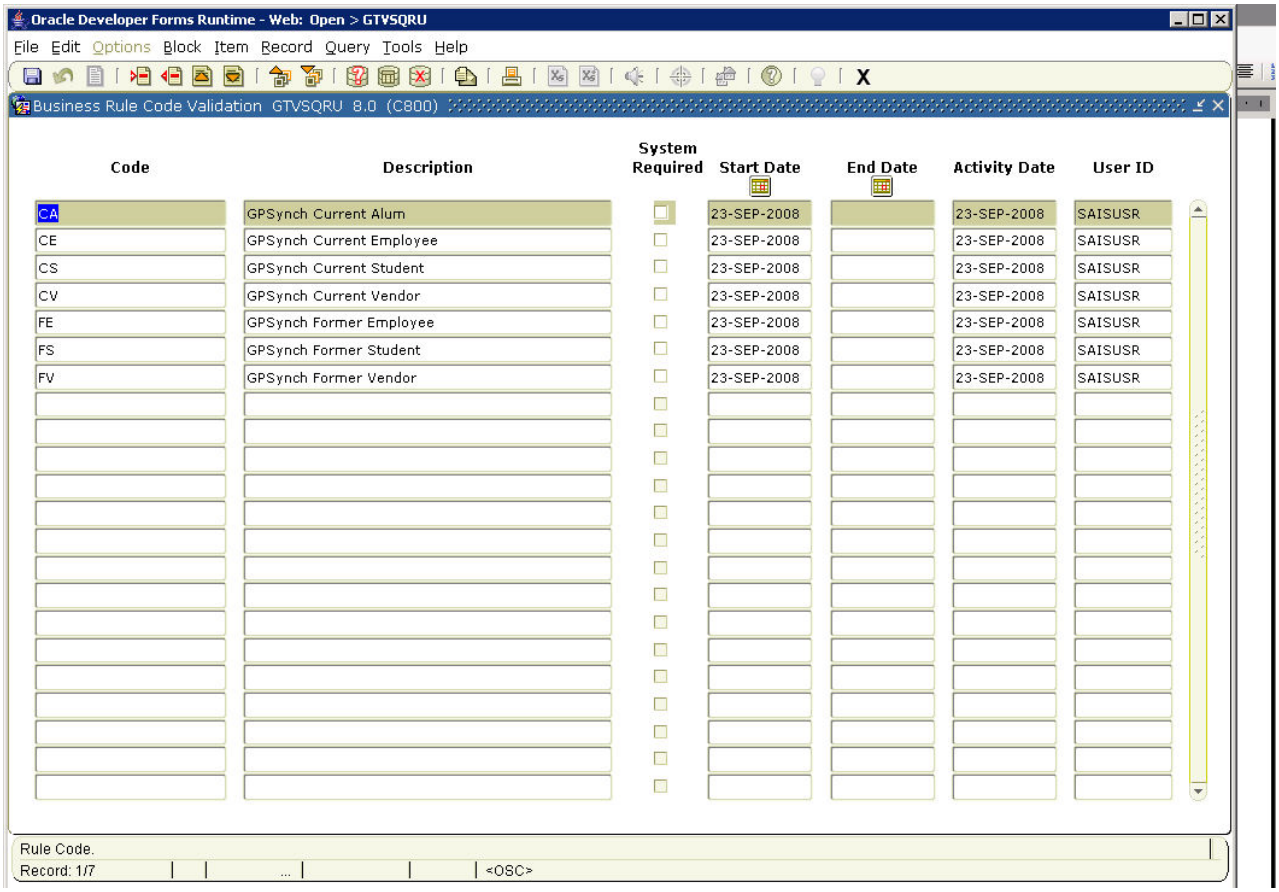
Business Rule Process Code Validation GTVSQPR 8.0 (SEED8B1)

Code	Description	System Required	Start Date	End Date	Activity Date	User ID
CARDHOLDER_ROLES	Cardholder roles	☑	11-SEP-2006		11-SEP-2006	GENERAL
ELEARNING	eLearning Integration rules	☑	04-JUN-2007		04-JUN-2007	BASELINE
HOUSING_ELIGIBILITY	Housing Integration, Eligibility Roles	☑	11-SEP-2006		11-SEP-2006	GENERAL
IAM	IAM Process Code	☐	27-DEC-2007		27-DEC-2007	GENERAL
INTCOMP	Integration roles	☑	11-SEP-2006		11-SEP-2006	GENERAL
SEVIS	SEVIS Processing	☑	02-OCT-2003		02-OCT-2003	GENERAL
VPD	VPD Processing	☑	12-SEP-2006		12-SEP-2006	GENERAL
GPSYNCH	GPSynch Big Bang enhancement	☐	23-SEP-2008	23-DEC-2008	23-SEP-2008	SAISUSR

FRM-40400: Transaction complete: 1 records applied and saved.

Record: 8/8 | ... | <OSC>

Next, we must define in Banner a Business Rule Code for each of the Systems current and former components of the Merge String, leave the SYSTEM REQUIRED unchecked, and default today's date for "Start Date" (see screen shot below):

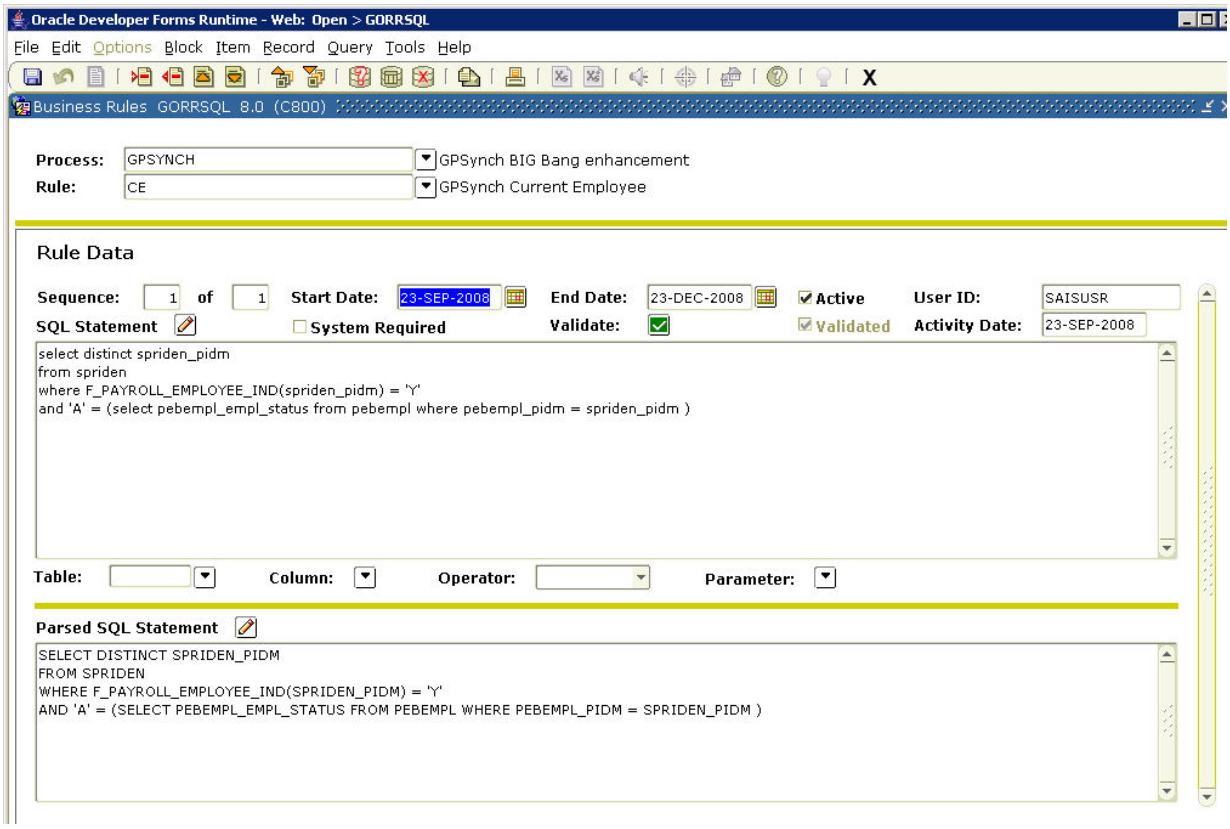


Code	Description	System Required	Start Date	End Date	Activity Date	User ID
CA	GPSynch Current Alum		23-SEP-2008		23-SEP-2008	SAISUSR
CE	GPSynch Current Employee		23-SEP-2008		23-SEP-2008	SAISUSR
CS	GPSynch Current Student		23-SEP-2008		23-SEP-2008	SAISUSR
CV	GPSynch Current Vendor		23-SEP-2008		23-SEP-2008	SAISUSR
FE	GPSynch Former Employee		23-SEP-2008		23-SEP-2008	SAISUSR
FS	GPSynch Former Student		23-SEP-2008		23-SEP-2008	SAISUSR
FV	GPSynch Former Vendor		23-SEP-2008		23-SEP-2008	SAISUSR

Now, we must define in Banner a Business Rule for each of the Business Rule codes we defined for this Business Process. Below are examples.

Using Banner Functions in RULE:
These statements are using the same functions that GPSynch is already utilizing, but these statements could be modified to look at whatever an Institution wishes to use as the determinate (see **Using Oracle External Tables** next). (see screenshots below) :

This rule is for a Current Employee that is Active.



This rule is for a Former Employee that is Active.

Oracle Developer Forms Runtime - Web: Open > GORRSQL

File Edit Options Block Item Record Query Tools Help

Business Rules GORRSQL 8.0 (C800)

Process: GPSYNCH GPSynch BIG Bang enhancement

Rule: FE GPSynch Former Employee

Rule Data

Sequence: 1 of 1 Start Date: 23-SEP-2008 End Date: 23-DEC-2008 Active User ID: SAISUSR

SQL Statement System Required Validate: Validated Activity Date: 23-SEP-2008

```
select distinct spriden_pidm
from spriden
where F_PAYROLL_EMPLOYEE_IND(spriden_pidm) = 'Y'
and 'A' != (select pebempl_empl_status from pebempl where pebempl_pidm = spriden_pidm)
```

Table: Column: Operator: Parameter:

Parsed SQL Statement

```
SELECT DISTINCT SPRIDEN_PIDM
FROM SPRIDEN
WHERE F_PAYROLL_EMPLOYEE_IND(SPRIDEN_PIDM) = 'Y'
AND 'A' != (SELECT PEBEMPL_EMPL_STATUS FROM PEBEMPL WHERE PEBEMPL_PIDM = SPRIDEN_PIDM)
```

SQL Statement compiled; parsed SQL updated.

Using Oracle External tables:

Since most Big Bang General Person conversions occur at the beginning of the implementation, Banner will not have enough data yet for GPSynch to determine the state of an individual in the system.

For example: General Person conversion has occurred, but the Vendors Table (FTMVEND) has not been loaded, nor has the Human Resources Employee Table (PEBEMPL), nor has the Student Tables (SARADAP, SGBSTDN, SFBTERM), nor has the Advancement Tables (APBCONS, organization, friends)… the process would need to gather the information from somewhere.

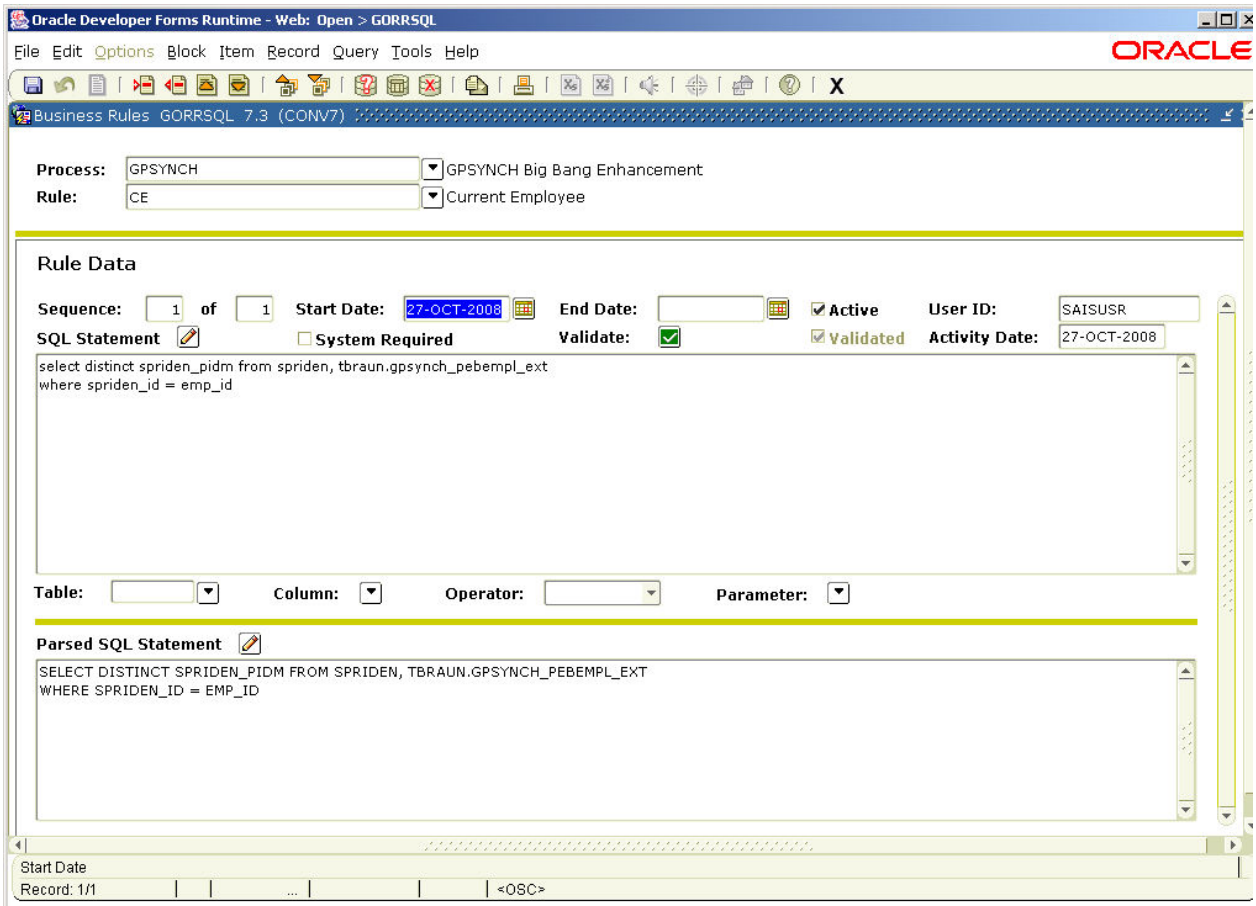
We can create Oracle external tables that read files of ID's pulled from each system. Each file would represent a type of individual from the system being pulled. Human Resources System would provide a list of ID's that are considered CURRENT EMPLOYEES (CE). Student Information System would provide a list of ID's that are considered CURRENT STUDENTS (CS), similarly for Finance and Advancement. (*files could be given for FORMER for each system as well)*.

Use of GORRSQL and external table:

```
create table gpsynch_pebempl_ext
(emp_id varchar2(9))
  ORGANIZATION EXTERNAL
   (TYPE ORACLE_LOADER
    DEFAULT DIRECTORY ext_merge_table_&&suffix
    ACCESS PARAMETERS
      (RECORDS DELIMITED BY NEWLINE
       FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""
       MISSING FIELD VALUES ARE NULL)
    LOCATION ('gpsynch_pebempl.csv')
    )
  REJECT LIMIT UNLIMITED;

create table gpsynch_ftvvend_ext
(vend_id varchar2(9))
  ORGANIZATION EXTERNAL
   (TYPE ORACLE_LOADER
    DEFAULT DIRECTORY ext_merge_table_&&suffix
    ACCESS PARAMETERS
      (RECORDS DELIMITED BY NEWLINE
       FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""
       MISSING FIELD VALUES ARE NULL)
    LOCATION ('gpsynch_ftvvend.csv')
    )
  REJECT LIMIT UNLIMITED;

create table gpsynch_sfbetrm_ext
(stud_id varchar2(9))
  ORGANIZATION EXTERNAL
   (TYPE ORACLE_LOADER
    DEFAULT DIRECTORY ext_merge_table_&&suffix
    ACCESS PARAMETERS
      (RECORDS DELIMITED BY NEWLINE
       FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""
       MISSING FIELD VALUES ARE NULL)
    LOCATION ('gpsynch_sfbetrm.csv')
    )
  REJECT LIMIT UNLIMITED;

create table gpsynch_apbcons_ext
(alum_id varchar2(9))
  ORGANIZATION EXTERNAL
   (TYPE ORACLE_LOADER
```

```
DEFAULT DIRECTORY ext_merge_table_&&suffix
ACCESS PARAMETERS
  (RECORDS DELIMITED BY NEWLINE
   FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY ""
   MISSING FIELD VALUES ARE NULL)
LOCATION ('gpsynch_apbcons.csv')
)
REJECT LIMIT UNLIMITED;
```

Sample Business Rules:



Oracle Developer Forms Runtime - Web: Open > GORRSQL

File Edit Options Block Item Record Query Tools Help

Business Rules GORRSQL 7.3 (CONV7)

Process: GPSYNCH — GPSYNCH Big Bang Enhancement

Rule: CS — Current Student

Rule Data

Sequence: 1 of 1 | Start Date: 29-OCT-2008 | End Date: | Active | User ID: SAISUSR

SQL Statement | System Required | Validate: | Validated | Activity Date: 29-OCT-2008

```
select distinct spriden_pidm from spriden, tbraun.gpsynch_sfbetrm_ext
where spriden_id = stud_id
```

Table: | Column: | Operator: | Parameter:

Parsed SQL Statement

```
SELECT DISTINCT SPRIDEN_PIDM FROM SPRIDEN, TBRAUN.GPSYNCH_SFBETRM_EXT
WHERE SPRIDEN_ID = STUD_ID
```

Start Date

Record: 1/1 | <OSC>

TROUBLESHOOTING

Receiving error “character buffer string too small”

CAUSE: The Oracle Directory no longer exists in the database.
ACTION: Redefine the Oracle Directory as the Oracle External table expects.

CAUSE: The Oracle Directory is no longer valid in the database or for OS.
ACTION: Redefine the Oracle Directory to the valid OS path.

CAUSE: The Oracle External Table Can’t be read.
ACTION: (a) Check to make sure the External table exists by doing a describe.
(b) Check that you can perform a SQL query of the external table.
(c) Check to ensure the CSV file is in the defined path for the Oracle directory, with appropriate permissions.
(d) Check to make sure the OS directory has appropriate permissions for LOG file and BAD file writing.
(e) who_wins.csv was FTP’d in wrong mode and now contains ^M. FTP in correct mode.
(f) if happens on SPBPERS conversion – then typically means spbpers_vetc_file_number in who_wins.xls accidentally got changed to spbpers_vetc_BANe_number - set back to correct column name

lengthy processing time:

CAUSE: Missing Indexes
ACTION: ensure necessary indexes exist and/or perform a rebuild of indexes

CAUSE: insufficient UNDO tablespace
ACTION: increase tablespace

CAUSE: insufficient tablespaces
ACTION: increase tablespace and/or have DBA perform sizing and tuning. See Action Line for FAQs on Oracle 10G tuning and new optimizer.

UPD_STATUS meanings:

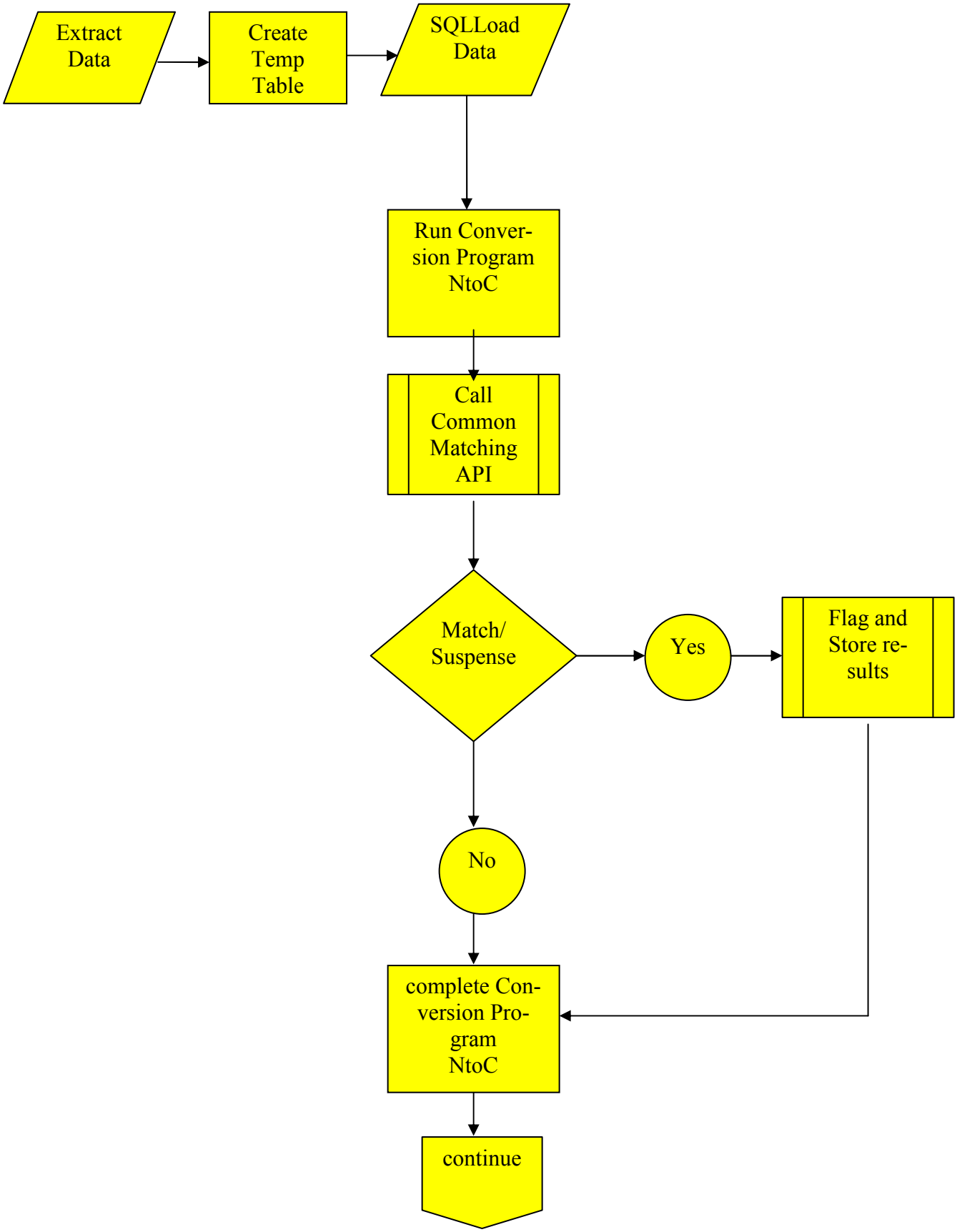
SPRIDEN_UPD_STATUS	
CODE	**MESSAGE**
ERR	an error has occurred when attempting to merge. Use the record ID to search CURCERR for the message and make corrections to data.
LEN	Overlength issues in Data need to be resolved
m	this record exactly matches the Banner record so the process simply flags the record as such - no actual merging occurs.
IBU	the incoming legacy record was inserted as a new current record and Banner current record is now an alternate record.
I	the incoming legacy record was inserted as is - no record in banner had to be updated - usually occurs for incoming name and id change records.
EIB	the incoming data did not exactly match any one record - however the information exists already in some form for the individual
IFL	the incoming legacy record was inserted as a change record as BAN won.

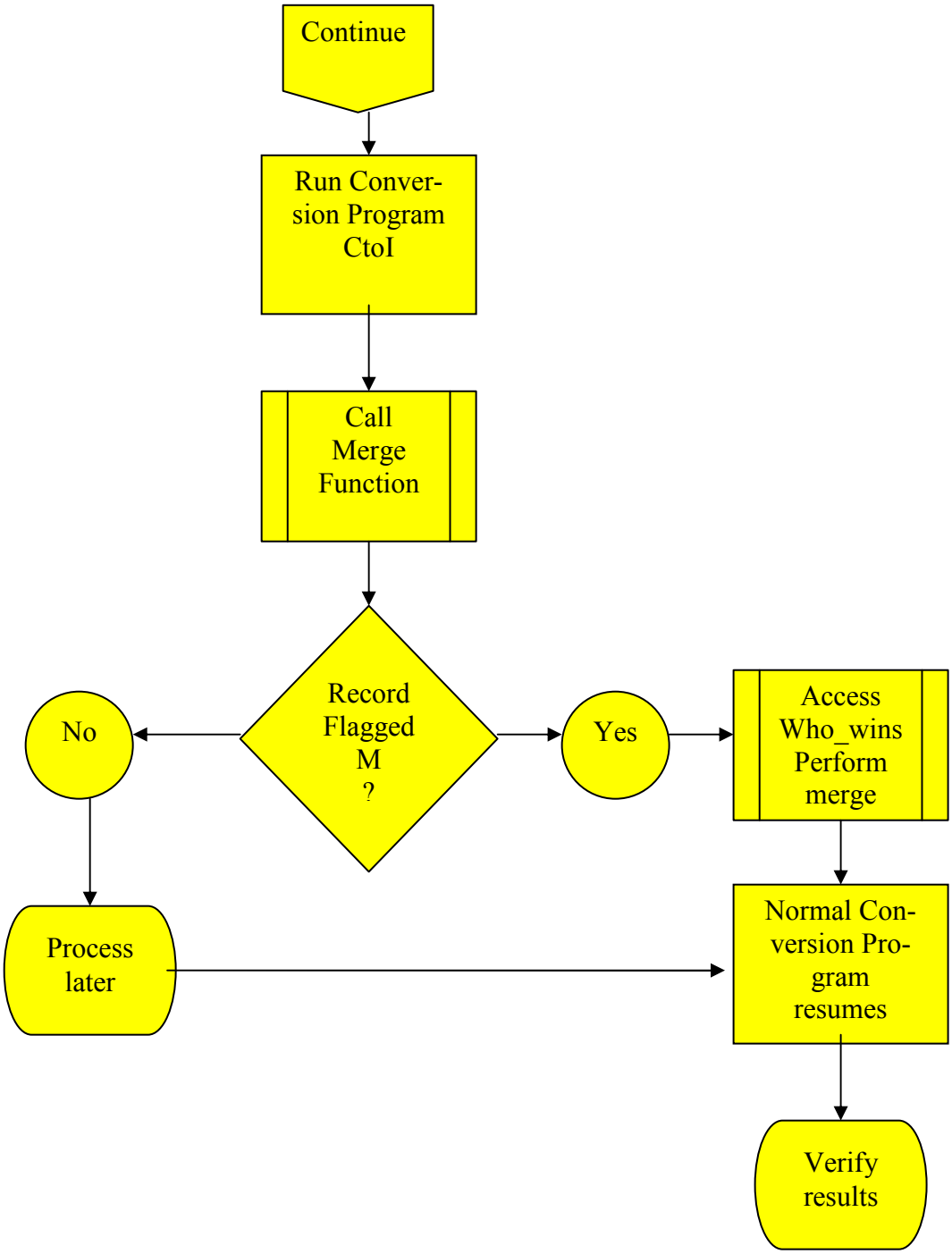
SPBPERS_UPD_STATUS	
CODE	**MESSAGE**
ERR	an error has occurred when attempting to merge. Use the record ID to search CURCERR for the message and make corrections to data.
LEN	Overlength issues in Data need to be resolved
BU	Banner record existed and was updated
I	Banner record did not exist so one was inserted – spbpers_merge_string and spbpers_who_wins will remain null

SPRADDR_UPD_STATUS	
CODE	**MESSAGE**
ERR	an error has occurred when attempting to merge. Use the record ID to search CURCERR for the message and make corrections to data.
LEN	Overlength issues in Data need to be resolved

m	this record exactly matches the Banner record so the process simply flags the record as such. no actual merging occurs, however the Banner record sequence number is stored to ensure that telephone numbers will be linked correctly.
BUs	Banner spraddr_status_ind was updated to the incoming legacy value. no actual merging occurs, however the Banner record sequence number is stored to ensure that telephone numbers will be linked correctly.
BAN	Banner spraddr_status_ind of matching address was not updated to the incoming legacy value as the who_wins rule was BAN.
BUI	Banner record was updated to be non-current for type (process uses end_date if can otherwise the status is set to inactive) and incoming record loaded under new sequence number. The Banner record sequence number is stored to ensure that telephone numbers will be linked correctly
I	Incoming record was inserted as is... no data contention with Banner.
SPRTELE_UPD_STATUS	
CODE	**MESSAGE**
ERR	an error has occurred when attempting to merge. Use the record ID to search CURCERR for the message and make corrections to data.
LEN	Overlength issues in Data need to be resolved
mBU	the incoming record telephone number was an exact match, however the incoming data had additional data to add to the record (status, preferred, comments, display on web)
mBL	the incoming record telephone number was an exact match but not linked to an address, however the incoming data had additional data to add to the record (status, preferred, comments, display on web)
I	the incoming record was inserted as is... no data contention with Banner. However, a new sequence number could have been generated and is stored in the temp table.

GOREMAL_UPD_STATUS	
CODE	**MESSAGE**
ERR	an error has occurred when attempting to merge. Use the record ID to search CURCERR for the message and make corrections to data.
LEN	Overlength issues in Data need to be resolved
mBU	the incoming record email address was an exact match, however the incoming data had additional data to add to the record (extension, unlisted, comments, international access)
I	the incoming record was inserted as is... no data contention with Banner.


Extract Data
Create Temp Table
SQLLoad Data
Run Conver-sion Program NtoC
Call Common Matching API
Match/ Suspense
Yes
Flag and Store re-sults
No
complete Con-version Pro-gram NtoC
continue


Continue
Run Conver-
sion Program
CtoI
Call
Merge
Function
Record
Flagged
M
?
No
Yes
Access
Who_wins
Perform
merge
Process
later
Normal Con-
version Pro-
gram
resumes
Verify
results