

USER MANUAL

BJS CCHRRD Conversion System

PRESENTED TO:
U.S. Department of Justice
Office of Justice Programs
Bureau of Justice Statistics

PRESENTED BY:
NORC at the
University of Chicago



at the UNIVERSITY *of* CHICAGO

BJS CCHRRD Conversion System

The MIT License (MIT)

Copyright © 2013 NORC at the University of Chicago

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Table of Contents

1 - Preface.....	1
1.1 Introduction.....	1
1.2 Requirements.....	3
1.3 Directory Structure	4
1.4 Input / Output.....	6
1.5 User Interface.....	8
1.6 Business Logic	11
1.7 Online Documentation.....	11
1.8 NORC Contact Information	11
2 – Installation.....	12
2.1 Installation Guide.....	12
2.2 Configuration	17
3 - System Usage	25
3.1 Overview	25
3.2 System Commands	29
3.3 Project Commands.....	35
4 - Business Logic	56
4.1 Business Logic Metadata	56
4.2 Business Logic – Coding Rules.....	6059
4.3 Pre-Process Rule Phase	64
4.4 Process Rule Phase.....	67
4.5 Rule Reference	732
5 – System Output	97
5.1 Relational Databases	97
5.2 SPSS Flat Files	97
5.3 Quality Control Report Files	98
5.4 Comparison Reports	100
5.5 Referential Integrity Report	100
5.6 Regular Expression Validation Report	101
5.7 Fatal Error Reports.....	103
Appendices.....	104

Appendix A: Standardized Database Schema

Appendix B: Recidivism Database Schema

1 - Preface

1.1 Introduction

The Conversion of Criminal History Record into Research Databases (CCHRRD) project was initially funded by the Bureau of Justice Statistics (BJS) at the Department of Justice to create a system capable of transforming US criminal history records into a standardized format suitable for recidivism research.

1.1.1 Project History

The initial funding from BJS supported the creation of research databases on a cohort of 75,000 prisoners released in 2005. The second round of funding added two additional cohorts, the US Federal Probationers cohort and the New York Juvenile Offenders cohort. All three cohorts were processed through the system to produce multiple research databases and these were delivered to BJS along with the current version of the software (v1.0), the business logic for all three cohorts, documentation including a user manual, and hands-on training.

With BJS's permission, the United States Sentencing Commission (USSC) has also funded a fourth cohort to be run through the CCHRRD Conversion System.

1.1.2 Project Mission

The purpose of the CCHRRD Conversion System is to provide an open source expert coding system for harmonizing parsed criminal history records obtained from the NLETS III system. To provide maximum flexibility, the Conversion System is licensed under the MIT open source license.¹

1.1.3 Development Status

Current Build:

The current version of the CCHRRD Conversion System, Version 1.0, represents the system's first production release, with functionality that represents the core features requested from BJS.

Development Build:

Version 1.02.a is currently under development at NORC and includes a new predictive modeling component capable of making intelligent predictions for offense codes that are not supported by the current business logic coding rules.

¹ For more information, visit <http://opensource.org/licenses/MIT>

Tentative Future Builds:

Version 1.10 will tentatively include the predictive modeling development improvements, along with an upgraded rule loading platform that standardizes the business logic coding rule creation process across all forms of crosswalks.

Version 2.0 will tentatively add a new web-based user interface for interacting with the conversion process, allowing users to run commands and update business logic without the need to use a command-line interface.

1.1.4 Change Log

Current Documentation Version: 1.1

1.0 to 1.1

- Major revisions
 - Appendix A
 - Appendix B

1.0

- Documentation Released

1.1.5 Definitions

Name	Description
<i>Cmd_param_short</i>	A command parameter short name, this will always have a hyphen followed by 1 character long, i.e. for the verbose parameter the cmd_param_short is -v
<i>Cmd_param_long</i>	A command parameter descriptive name, this will always have two hyphens followed by a descriptive name, i.e. for the verbose command the cmd_param_long is --verbose
<i>Demo</i>	The name of a new conversion project used as an example throughout the user manual document
<i>Highest_round</i>	The most recent round of output from a process command, i.e. if the user has run two rounds of processing the highest round will be round2
<i>INSTALLDIR</i>	Refers to the directory path where the conversion software was installed. The user has the option of setting the install directory during step 4 of the installation process as described below. The default location for the <i>INSTALLDIR</i> is: C:\Program Files (x86)\BJS CCHR
<i>Phase</i>	A dynamic placeholder for a phase, supported phases are stage, pre_process, process, transform, load_recid_db, and load_recid_flat
<i>Program data</i>	The directory selected by the user to store project files
<i>PROJECTDIR</i>	The projectdir is located within the program data directory and will be named after the project
<i>Region</i>	A dynamic placeholder for this document indicating that region should be replaced with any supported region, ie al, ak, az, etc

<i>ROOTDIR</i>	See program data
Round	A placeholder indicating a conversion round, i.e. round1, round2 and so forth
Segment	A placeholder indicating the name of a segment; segments are subject, arrest, court-sentence, supervision, and demographic.
SYSTEMOUTDIR	Refers to the directory path where output files file be saved to
Timestamp	A placeholder indicating some date time
Working Project	The current project that the system will run commands on. Use activate to set working project, there can only be one working project in the system
offense flag variables	The offense flag variables for the arrest segment are ancic, achg, ainc, adom, aweap, acdv, apg and cncic, cchg, cinc, cdom, cweap, cc dv, cpg for the court-sentencing segments
Sentencing variables	The sentencing variables are cnf, incmax, incmin, suspmax, suspmin, and prb

1.2 Requirements

This guide assumes that you are installing this software on a dedicated application server that meets the following requirements:

1.2.1 Hardware

In order to optimize performance of the CCHRRD Conversion System, NORC recommends the following minimum hardware configuration:

- 16 GB of RAM (32 GB preferred)
- 200 GB Hard Drive (or read/write access to a 200+ GB network drive)
- Write access to the directory where the software will be installed
- At least one multi-core, high speed Intel or AMD processor (4 preferred)

Increased RAM, hard drive space, processor speed, or a greater number of processors may improve performance.

1.2.2 Software

The BJS CCHRRD Conversion System has the following application dependencies:

- Microsoft Windows Server 2008 R2²
- Python 2.7 development environment and runtime libraries
- NORC Python-ETL open source library

² The software may work on other versions of Windows but it has not been tested

- 7-Zip file archiver
- A MySQL 5.1 - 5.5 database server for exclusive use by this system
- Mercurial, Git or Apache Subversion source code control system (optional)
- SAS 9.X (required for other version of windows)

1.2.3 Security

The end user should expect to store the data, both raw and converted, on a password-protected, role-based network drive. Likewise, the application's server and its file structure should follow secure, least-privileges best practices. If a separate database server is used, its file structure should also follow the same practices. The entire system should be firewalled to prevent the possibility of hostile intrusion, and external access points should also be secured. To eliminate the chance of malware spreading or unwarranted penetration, NORC recommends granting entry to the various environments through VPN, Citrix, or some other remote login and desktop emulation method. Finally, we recommend installing MySQL on the application server and disabling TCP/IP connections to the database, forcing all connections to be made through a local client.

1.3 Directory Structure

Before proceeding with installation, it will be helpful to understand how the BJS CCHRRD Conversion System is structured. After the installation process is complete, a folder named **BJS CCHR** will be created in the selected installation directory.

The **BJS CCHR** folder consists of a number of directories, the most important of which are described below:

INSTALLDIR\BJS CCHR

```
|--- bin
|--- business logic
|--- cchr
```

The **bin** directory contains the executable command files. All commands will be issued from this directory.

The **cchr** directory contains the software source code (**cchr\src**), including the static recidivism conversion rules, which can be found in the **cchr\src\recid_rules_db** and **cchr\src\recid_rules_flat** folders.

The **business logic** directory contains business metadata and business coding rules for three previous projects, the Federal Probationers cohort, the Juvenile Offenders cohort, and the 2005 Prisoners Release cohort:

```
BJS CCHR
|--- business logic
|--- Federal Probationers
|--- Juvenile Offenders
|--- Prisoners Release in 2005
|--- metadata
|--- rules
```

After the installation process is complete, the user will be required to start a new project. Upon issuing the **start_project** command, a new project file structure is created. In the example below, the user has created a new project called *demo*:

```
PROJECTROOT
|--- build
|--- metadata
|--- rules
|--- project_data
|--- demo (PROJECTDIR)
|--- input
|--- metadata
|--- output
|--- predict
|--- rules
|--- tmp
```

The **build** directory contains the business logic to be used as the base set of rules on all new projects. The build directory is created during software initialization, and the user is allowed to select which previous project to use as the base set of rules.

The **project_data** directory contains all the project files for CCHRRD conversion projects. Each project is assigned its own folder, and in the example above there is one project called *demo*.

The **input** directory is the default location in which software will search for the NLETS intermediate files and the project sample file.

The **metadata** directory contains all of the project's business logic metadata.

The **output** directory contains all report and SPSS flat files created during the conversion process.

The **predict** directory is not supported in the current version (1.0), but will contain files related to CCHR predictive models in future versions.

The **rules** directory contains the business logic coding rules and supporting files.

The **tmp** directory contains system output files related to the success or failure of each command issued by the program. The files created here can be useful for system diagnostics and troubleshooting.

1.4 Input / Output

The following input files are required by the BJS CCHRRD Conversion system:

1. An SPSS subject file containing one row for each subject within the study, with the following columns:
 - ▶ Casenum
 - ▶ State of release
 - ▶ Month of birth
 - ▶ Day of birth
 - ▶ Year of birth
 - ▶ Month of release
 - ▶ Day of release
 - ▶ Year of release
2. Intermediate parsed data files from the NLETS RAP sheet parsing system. The NLETS system extracts all criminal history records across all 50 states from the Interstate Identification Index and transforms these records into bar delimited files. The NLETS system creates multiple parsed files, which we will refer to as **segment** files. The following segment files are required:
 - ▶ **Arrest segment:** contains one arrest record per row
 - ▶ **Sentence segment:** contains one sentence record per row
 - ▶ **Demographic segment:** contains one demographic record per row

Note: while the NLETS system also creates a supervision segment, this segment is not currently supported in the BJS CCHRRD Conversion System.

After creating a new project, the input files must be placed in the **PROJECTDIR/input** directory. After receiving the files from NLETS, it is important to review these files to verify that they conform to the

specifications detailed in the business logic metadata (see section 4.1 for more information on reviewing business logic metadata.)

The BJS CCHRRD Conversion system creates the following types of output: flat file output, database output, temporary file output, and command line system output.

1) Flat file output

The following flat files are created during use of the system:

- ▶ **SPSS standardized relational database extract files** are created from the segment tables after the **Process** phase is complete (see section 3.3.3 for more details on the Process phase). These files are saved to the *PROJECTDIR/output/review/SET/SEGMENT/REGION/* folder.
- ▶ **SPSS recidivism database extract files** are created from the arrest and sentence segment tables after the **Recidivism Database Load** phase is complete (see section 3.3.5 for more details on the Recidivism Database Load phase). These files are named after the segment name, e.g. “arrest” or “sentence.” Additionally, all of the records that were removed from these segments based on the recidivism coding rules are saved to SPSS files, in order to allow experts to manually review the coding process in detail. Each of these files are named *segment_del_records*, and can be found in the *PROJECTDIR/output/transform* folder.
- ▶ **The SPSS Criminal History Flat, Criminal History Summary, and Demographic Flat files** are created after the **Recidivism Flat File Load** phase is complete (see section 3.3.6 for more details on this phase). These files are saved to their respective directories in the *PROJECTDIR/output/recid* folder.
- ▶ **Log files** are created for each command execution, and provide information on any controlled system error. For example, if an arrest record does not match any offense coding rules, details about the coding failure are logged and saved to the *PROJECTDIR/log/SET/COMMAND* folder.
- ▶ **Coding rule report files** are created based on the log output for the **Process** phase. The reports provide information on any errors or warnings detected during the application of the business logic process coding rules. These files are saved to the *PROJECTDIR/output/review/SET/SEGMENT/REGION/ROUND* folder.
- ▶ **Difference report files** are created after running the **Compare** command (see section 3.3.9 for more details on the Compare command), and are saved to the *PROJECTDIR/output/review/SET/SEGMENT/REGION/ROUND* folder.

2) Database output

Two relational MySQL databases are created after the **Stage** command has completed. The **standardized relational database** is finalized after the **Process** phase and the **recidivism relational database** is finalized after the **Recidivism Database Load** phase. These databases can be accessed via the MySQL visual studio tool, the MySQL command line tool, or by using an ODBC connection.

3) Temporary file output

A number of **temporary files** are created detailing the success or failure of each command. In addition, some commands will use the temporary files to act as a temporary data store. These files are saved to the **PROJECTDIR/tmp** folder.

4) Command-line system output

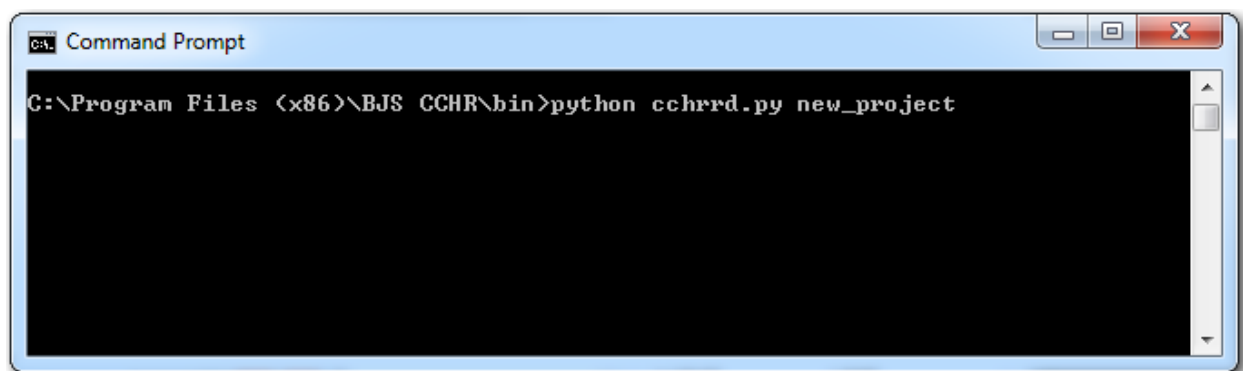
After issuing a command, the system will output text to the command prompt window, detailing the status of the current operation.

1.5 User Interface

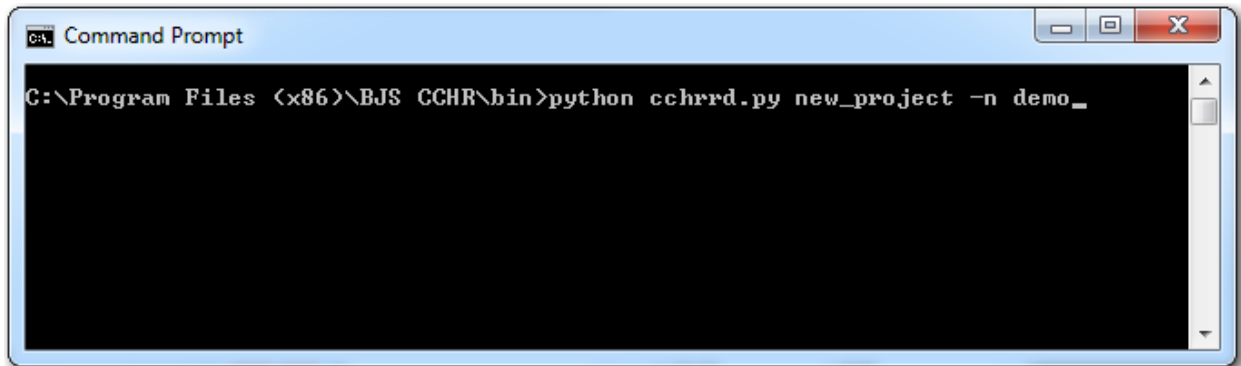
For version 1.0 of the BJS CCHRRD Conversion system, users interact with system commands using a command-line interface and a configuration file, and interact with the business logic using a series of Excel, text, and Python files.

1.5.1 Command-line interface

The **command-line user interface** requires the user to execute commands using the Windows cmd prompt. The majority of the commands are executed using the **cchrrd.py** module. An example of running the **New Project** command can be seen below:

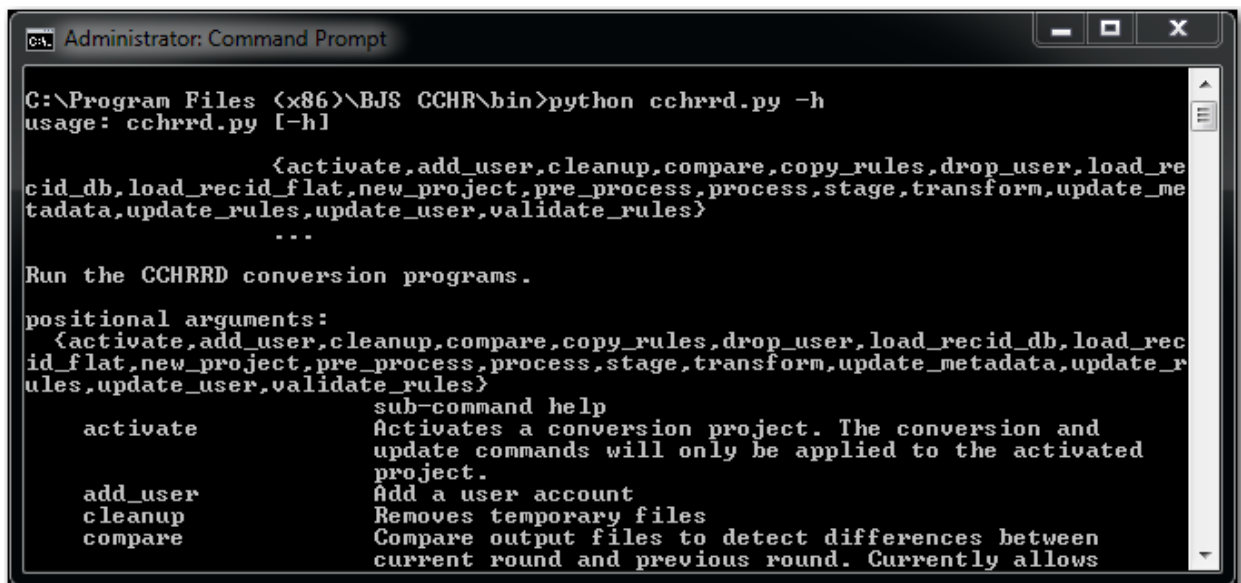


Command-line parameters are also accepted by most of the CCHRRD Conversion System commands. Parameters are passed using either short hand notation with one dash (i.e., `-cmd_param_short`) or by passing the full command parameter name with two dashes (i.e., `--cmd_param_long`). In the example below, a user is about to start a new project using the **new_project** command, and has specified the name “demo” for the project using the shorthand “-n” notation, although they could also do the same using the full parameter notation, “--name”:



```
Command Prompt
C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py new_project -n demo_
```

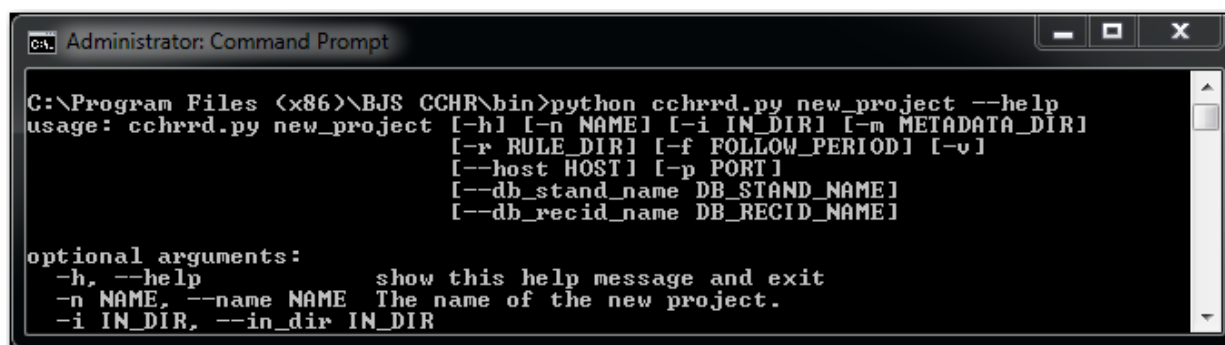
To see a full list of supported parameters along with details about the command, you can pass the “-h” or “--help” parameter with any command.



```
Administrator: Command Prompt
C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py -h
usage: cchrrd.py [-h]
                  {activate,add_user,cleanup,compare,copy_rules,drop_user,load_re
cid_db,load_recid_flat,new_project,pre_process,process,stage,transform,update_me
tadata,update_rules,update_user,validate_rules}
                  ...

Run the CCHRRD conversion programs.

positional arguments:
  {activate,add_user,cleanup,compare,copy_rules,drop_user,load_recid_db,load_rec
id_flat,new_project,pre_process,process,stage,transform,update_metadata,update_r
ules,update_user,validate_rules}
    activate           Activates a conversion project. The conversion and
                        update commands will only be applied to the activated
                        project.
    add_user           Add a user account
    cleanup            Removes temporary files
    compare            Compare output files to detect differences between
                        current round and previous round. Currently allows
```



```

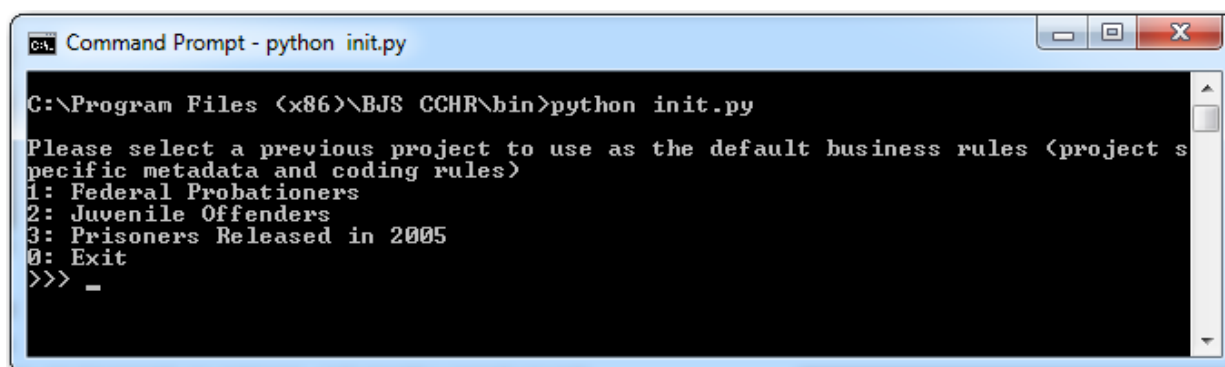
Administrator: Command Prompt

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py new_project --help
usage: cchrrd.py new_project [-h] [-n NAME] [-i IN_DIR] [-m METADATA_DIR]
                             [-r RULE_DIR] [-f FOLLOW_PERIOD] [-v]
                             [--host HOST] [-p PORT]
                             [--db_stand_name DB_STAND_NAME]
                             [--db_recid_name DB_RECID_NAME]

optional arguments:
  -h, --help            show this help message and exit
  -n NAME, --name NAME  The name of the new project.
  -i IN_DIR, --in_dir IN_DIR

```

With the exception of the **System Initiation** command and use of the help parameter, the user will be required to provide their username and password when invoking a command. After entering in their account credentials, the command will either execute or it will prompt the user for additional directions. Below is an example of the **System Initiation** command, which prompts the user to select an existing project to use as the default source of business rules for new projects:



```

Command Prompt - python init.py

C:\Program Files (x86)\BJS CCHR\bin>python init.py

Please select a previous project to use as the default business rules (project s
pecific metadata and coding rules)
1: Federal Probationers
2: Juvenile Offenders
3: Prisoners Released in 2005
0: Exit
>>> -

```

After a command is finished running, you will be returned to the command line and will be allowed to run new operations. At any point, a running operation can be terminated by pressing **Ctrl+C**, but terminating the command in this manner may result incomplete conversions.

1.5.2 Business Logic Interface

There are two methods of updating the business logic. The first method involves updating the **Microsoft Excel crosswalk files**. These files were created in an effort to provide a commonly known interface to allow for common rule updates. The Excel rule files may be used to update the **pre-process ORI/FED rules**, the **process offense rules**, and the **process sentence rules**. These rule files can be found in the *PROJECTDIR/rules/ori_crosswalks*, *PROJECTDIR/rules/offense_crosswalks*, and

PROJECTDIR/rules/sentence_crosswalks folders, respectively. Please see chapter 4 for more information on working with the Excel rule files.

The second method for updating rules involves making changes directly to the business coding rules and modifying the **rule controllers** (func.py files) and/or the **rulesets** (the machine readable files created by the crosswalk files). These updates should only be completed by those with at least a basic knowledge of the Python programming language and a proficient understanding of the BJS CCHRRD Conversion system.

1.6 Business Logic

At the core of the BJS CCHRRD Conversion system is the **business logic**. The business logic is composed of **metadata and coding rules**. The metadata files provide the system with the specifications of the **NLETS intermediate files**, the two **relational databases**, the **Criminal History Flat SPSS file**, the **Criminal History Summary SPSS file**, and the **Demographic Flat SPSS file**.

The core logic behind the coding rules was developed by a team of researchers at NORC through years of collaboration with representatives from BJS and state and federal criminal justice agencies. This business logic instructs the system on how to process the NLETS intermediate files from a disjoint set of state and federal records into a standardized database, and then transforms this database into a recidivism research database and recidivism research flat files. As the business logic was defined, it was translated into **rule controllers** (Python modules), **rulesets** (crosswalks), and **individual coding rules**. Each rule controller uses one or more rulesets, with each ruleset representing a set of one or more coding rules. Currently there are 265 rule controllers, 1,781 rulesets, and over 1.7 million coding rules.

1.7 Online Documentation

Detailed online documentation can be found at <http://cchrrd.readthedocs.org/en/latest/>. This is living documentation, and is subject to updates as the software grows. The latest documentation branch is for Version 1.0, but documentation for all subsequent versions will also be made available on the above mentioned site as they are released.

1.8 NORC Contact Information

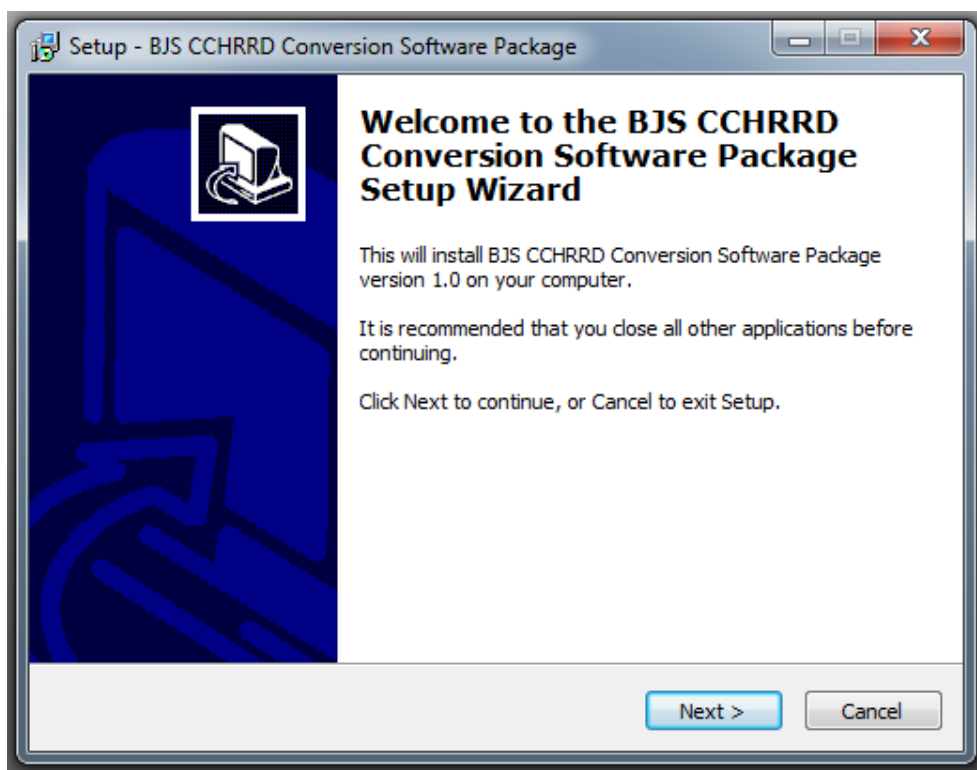
2 – Installation

2.1 Installation Guide

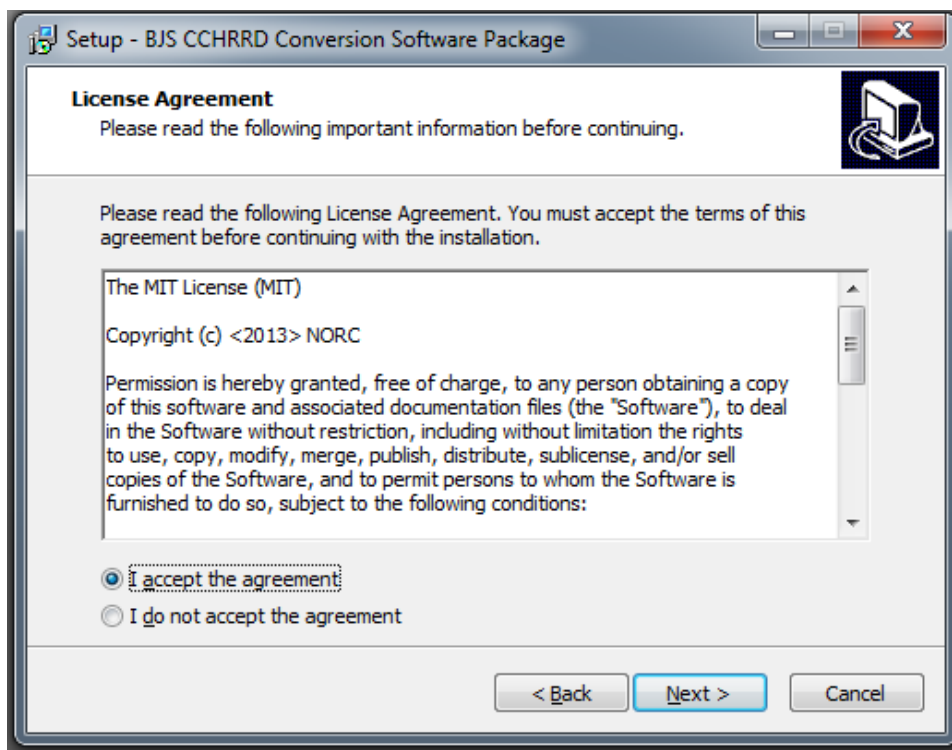
This is a guide to the BJS CCHRRD Conversion System installation process.

Installation Steps

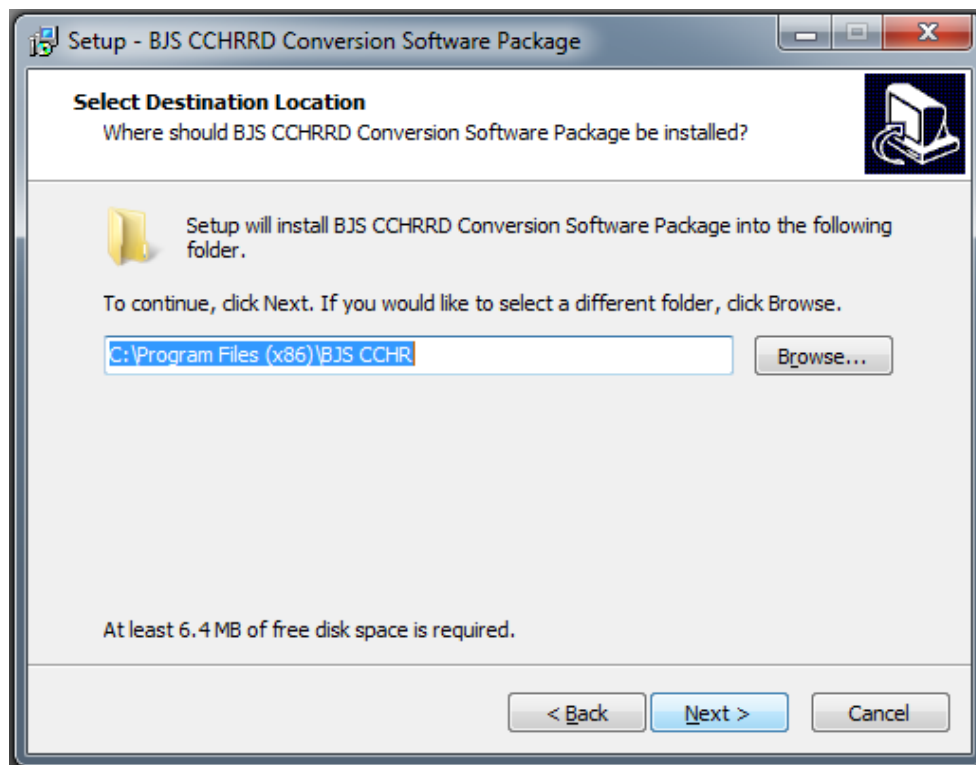
- 0) Before beginning installation, please verify that the hardware and software requirements have been met, as described in section 1.2.
- 1) Enter the installation DVD into the DVD drive. An installation wizard should automatically start, but if it does not, then navigate to the DVD drive and open the **install.exe** file on the DVD.
- 2) Click **Next** on the below screen



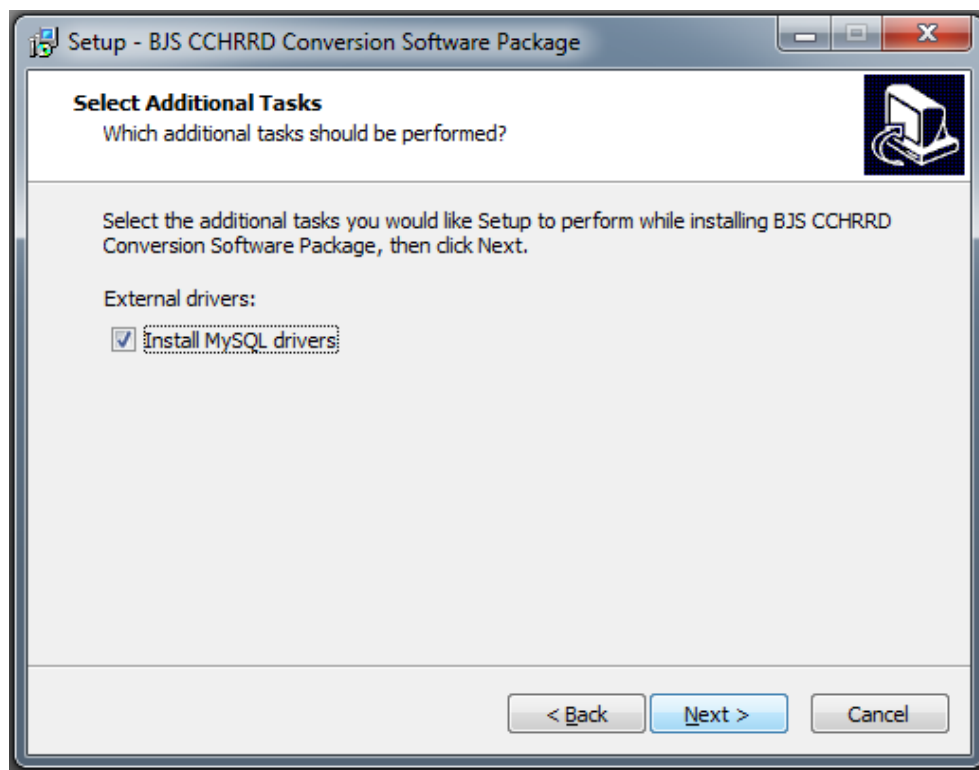
3) Accept the user agreement and click **Next**



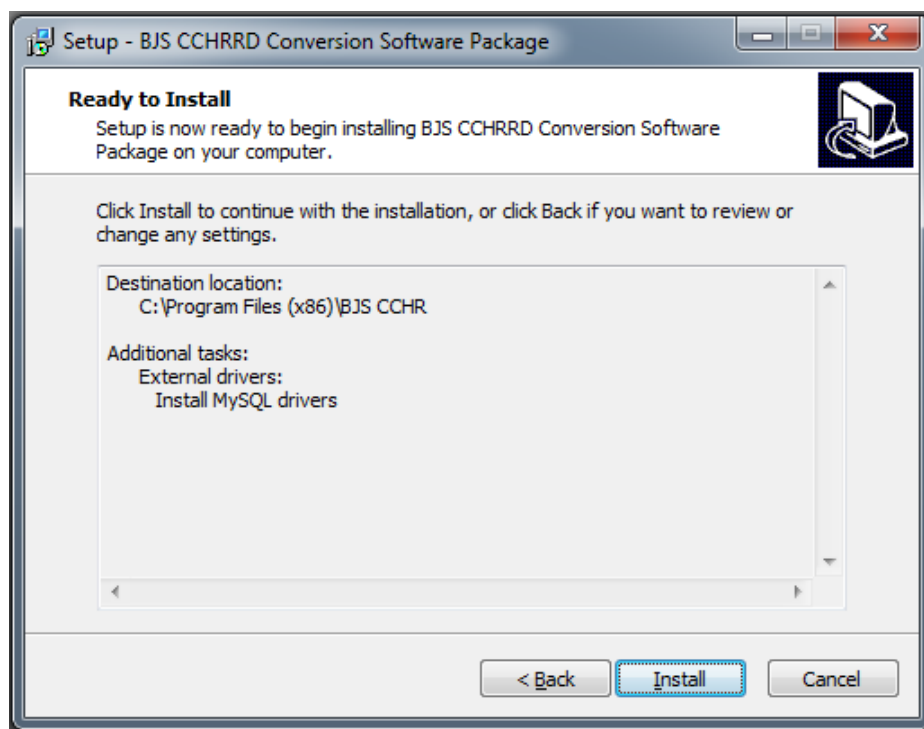
- 4) Accept the default installation location, or specify a new installation location and click **Next**.



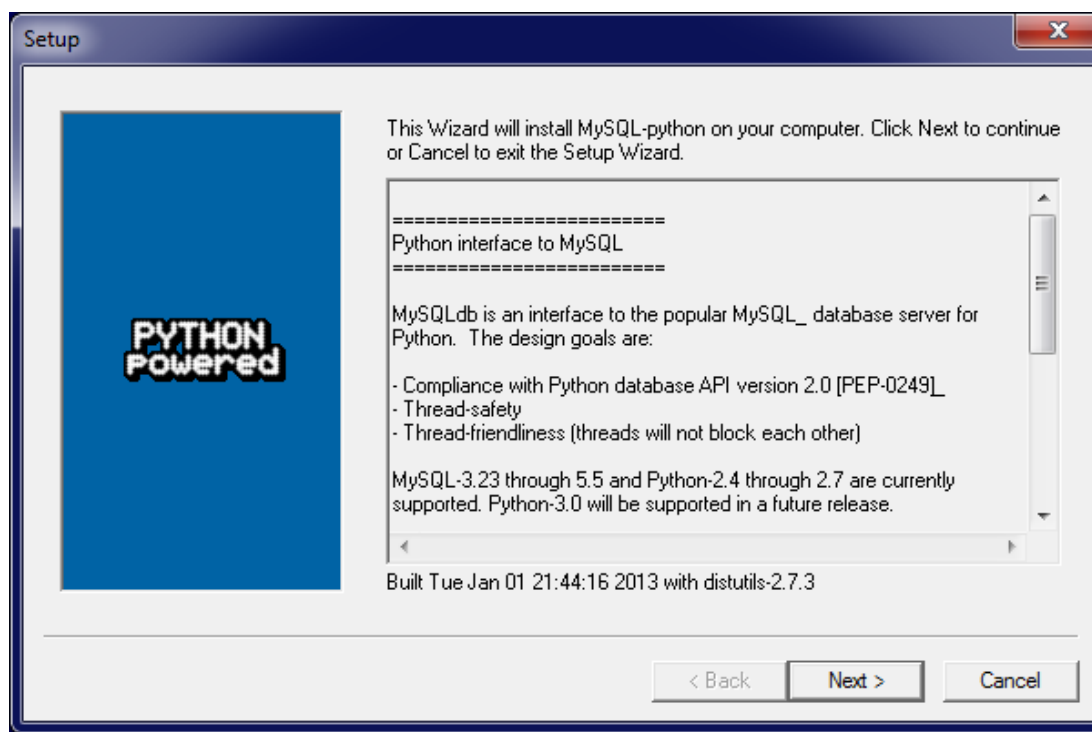
- 5) Make sure to keep the **Install MySQL Drivers** option selected and click **Next**.



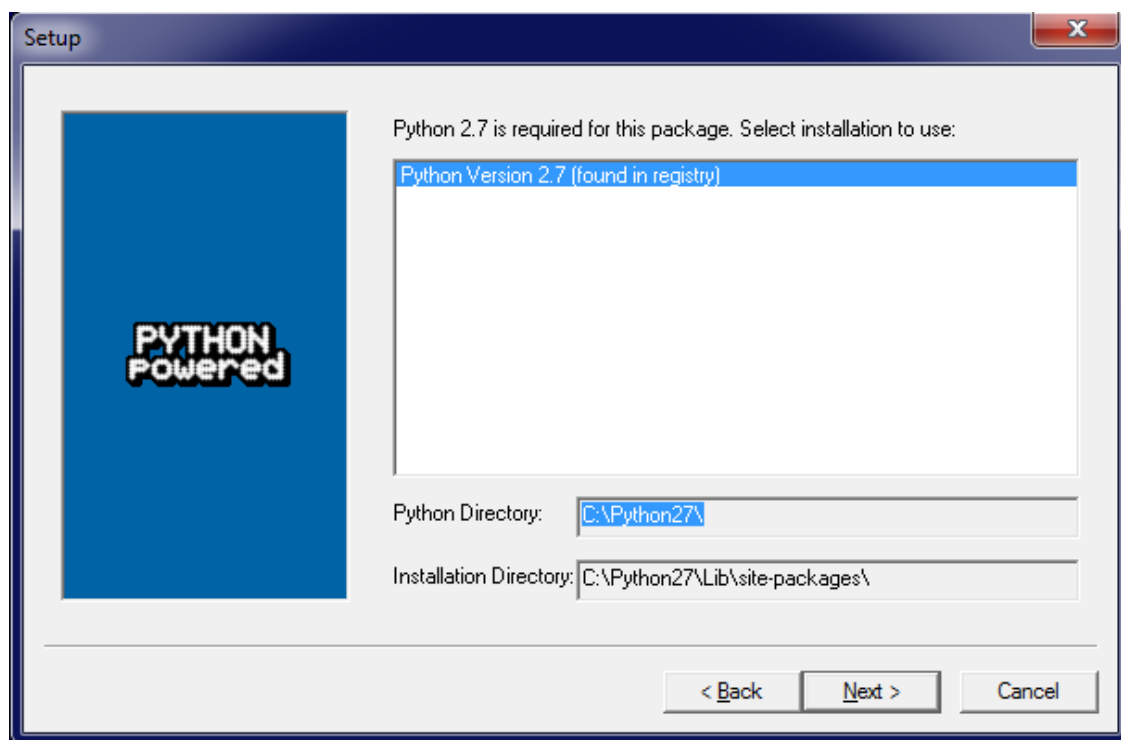
6) Click the **Install** button.



7) Click the **Next** button on the MySQL Python Driver setup page.

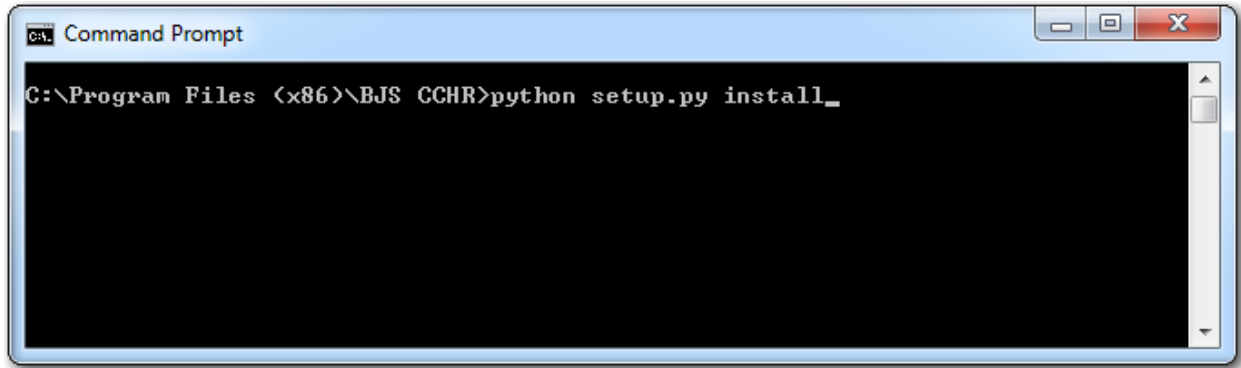


- 8) Choose the Python 2.7 directory for installation and then click **Next** on this screen and **Next** or **Finish** on the following screens.



- 9) After the installation setup is finished, open a command prompt (cmd) and navigate to the installation folder (**INSTALLDIR**).

Once in the installation folder, issue the following command and press ENTER: **python setup.py install**



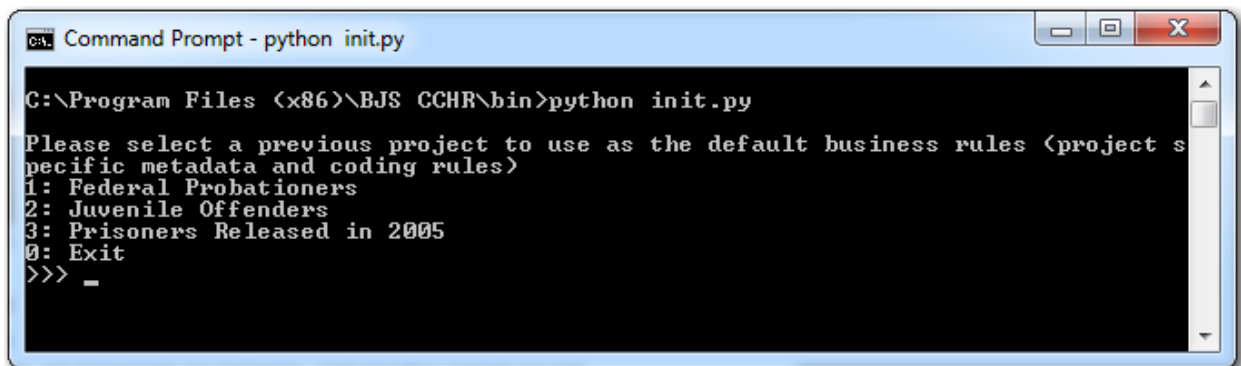
This will install all dependencies necessary for the program to run. After the setup has finished installing, the software will be ready for configuration.

2.2 Configuration

After the installation has been completed, it may now be configured to suit the needs of your projects. There are two phases of configuration: **system configuration** and **project configuration**. System configuration prepares the software for use, and project configuration creates and prepares a criminal record conversion project.

2.2.1 System Configuration

- 1) Open a command-line prompt, navigate to the project bin directory (**INSTALLDIR\bin**), and issue the init command by typing **python init.py**



You will be prompted to select a past project from which to load the default business rules.³ Enter the number that corresponds to the project business rules you want to use as the default rules.⁴

- 2) After selecting the source project, you will be prompted to choose a location for the new project's data directory. The **project data directory** will contain one project folder to store the configuration, rules, input, output, and log files for each project. Please see section 1.3 for more details on the project folder.

```

C:\Program Files (x86)\BJS CCHR\bin>python init.py

Please select a previous project to use as the default business rules (project s
pecific metadata and coding rules)
1: Federal Probationers
2: Juvenile Offenders
3: Prisoners Released in 2005
0: Exit
>>> 2

Please enter the directory where project data will be stored
>>>

```

Note: It is **required** that the user running the program has read/write access to the project data directory. It is **recommended** that you choose a directory that is encrypted and has limited access to only the users that will be running the conversion system.

- 3) After entering the project directory, the user will be prompted to enter the host address and port for the MySQL Community Server.

```

C:\Program Files (x86)\BJS CCHR\bin>python init.py

Please select a previous project to use as the default business rules (project s
pecific metadata and coding rules)
1: Federal Probationers
2: Juvenile Offenders
3: Prisoners Released in 2005
0: Exit
>>> 2

Please enter the directory where project data will be stored
>>> d:\bjs data

Please enter the database host (default=localhost)
>>>

Please enter the database port (default=3306)
>>>

```

³ This list is based on the projects that are in the business_rules folder (*INSTALLDIR\BJS CCHR\business_rules*). If you want to use a different project, place the business rules for that project in the business_rules folder.

⁴ As of 10/2013 the Juvenile Offender rules reflect the most recent NLETS parsing logic

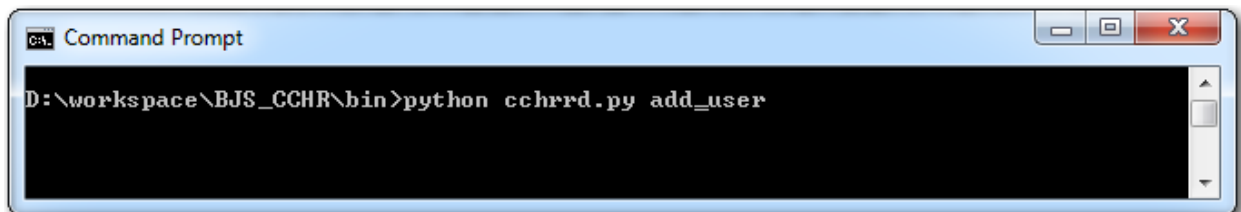
Both of these options have defaults set for a local client connection to the MySQL Community Server installation (as recommended in section 1.2.3.) To accept the default value, simply press **ENTER** without entering any text.

After choosing the MySQL port, the system configuration will proceed to copy the business logic of the desired project to the system output build directory and will create the system configuration file (*INSTALLDIR\cchr\src\bjs.config*)

2.2.2 Project Configuration

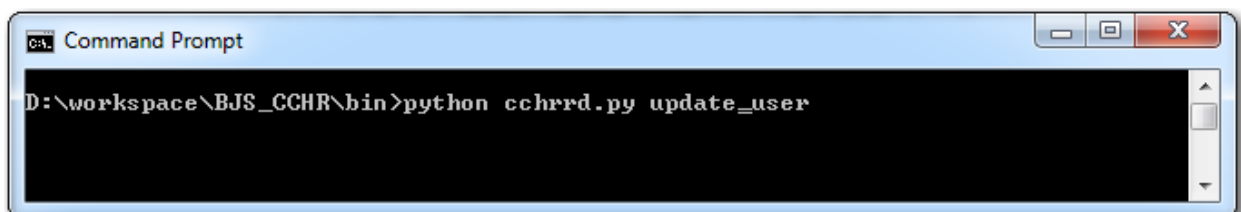
Account Management

After the system configuration has been completed, you will then be able to issue commands to the CCHRRD Conversion System. All commands require a username and password; the initial user will therefore need to know a username and password that has root privileges to the MySQL server. At this point, the initial user should create accounts for all other users. To do this, the user issues the **add_user** command and follows the command-line prompts.



```
Command Prompt
D:\workspace\BJS_CCHR\bin>python cchrrd.py add_user
```

After new user accounts have been created, multiple users can use the system. However, only one user should run a given project at a particular time in order to avoid data writing conflicts. If user accounts need to be updated or removed, use the **update_user** and **drop_user** commands (see sections 3.2.3 and 3.2.4 for more details).



```
Command Prompt
D:\workspace\BJS_CCHR\bin>python cchrrd.py update_user
```



Project Creation

To create a new project, navigate to the *INSTALLDIR\bin* folder and execute the **new_project** command.



You will be prompted to enter the **project name** and to enter **the number of years for the follow-up window**. The follow-up window specifies the number of years following a prisoner's date of release for which recidivism statistics will be calculated for that prisoner.

The **new_project** command also accepts a number of optional command-line input parameters to allow the user to change system default values at the time of project creation. For example, the user can pass the **-n** parameter to specify the name of the project, and so forth. Please see section 3.2.2 for full list of supported parameters.

Project Settings

After the new project command has finished running, a project directory will be created in the *PROJECTROOT* folder. Within the *PROJECTDIR* directory there is a project configuration file called "[PROJECT_NAME].config". The project configuration file should be reviewed after creating a new project to verify that the project settings were all successfully created.

The **project configuration file** has a number of settings which determine how the conversion system operates on a given project, which the user may modify to suit their project needs. The system creates the configuration file with the most appropriate settings given the parameters specified in the **new_project** command, and because of this the user may never need to change any of the options in the configuration file. However, it is a good practice to review the file after creating a new project, which can be accomplished by opening the file in a text editor. The configuration file is split into a number of sections:

default, security, debug, etc. Sections can be identified by `[section_name]`. Each section contains one setting per line following the format `setting_name = setting_value`. See below for more information about the various settings in the configuration files.

2.2.3 Project Configuration File

[default]

The default section contains the most commonly used configuration options. The majority of these options are directories allowing the user to change where the system looks to load and/or save files.

Name	Description	Default
root_dir	Specifies the location of the root project directory	[ROOTDIR]
project_dir	Specifies the location of the project directory	[PROJECTDIR]
in_dir	Specifies the directory containing the NLETS intermediate parsed files and sample file	[PROJECTDIR]\input
out_dir	Specifies the directory for output files (reports, database flat file extracts, SPSS flat files, etc.)	[PROJECTDIR]\output
tmp_out_dir	Specifies the directory for saving temporary system output files (system log files, debug files, etc)	[PROJECTDIR]\tmp
log_dir	Specifies the directory for saving conversion command log files	[PROJECTDIR]\output\log
report_dir (deprecated)	Specifies the location for saving conversion report files (files now saved to review_dir)	[PROJECTDIR]\output\report
review_dir	Specifies the location for saving review files, these include report files and SPSS review files	[PROJECTDIR]\output\review
meta_dir	Specifies the directory containing the business logic metadata	[PROJECTDIR]\metadata
rule_dir	Specifies the directory containing the business logic coding rules and supporting files	[PROJECTDIR]\rules
offense_rules	Specifies the directory where the offense rules are located	[PROJECTDIR]\rules\process_rules_2
admrec_dir	Specifies the directory where the administrative rules are located	[PROJECTDIR]\rules\process_rules_3
xwalk_out_dir (deprecated)	Specifies the directory for crosswalk output (files now saved to rule_dir/crosswalk_raw)	[PROJECTDIR]\output\crosswalk
stable_dir (deprecated)	Specifies the directory for stable rule output	[PROJECTDIR]\output\stable
predict_dir	Specifies the directory for prediction model output (supported in version 1.1+)	[PROJECTDIR]\predict
follow_period	Specifies the number of years following a given prisoner's day of release for which the software will calculate recidivism statistics	5
current_set	The current set of intermediate data files	1
cores	Specifies the cores the software should use when running parallel commands	2
transform_threads	The number of threads to use when running the recidivism transformation conversion	100

[security]

The security section stores settings for distributed computing security. Currently, this section has no impact on the system as distributed computing is not supported in version 1.X

Name	Description	Default
HMAC_KEY (unsupported)	Specifies the md5 key to use for distributed processing (unsupported in version 1.X)	d41d8cd98f00b204e9800998ecf8427e

[debug]

The debug section allows the user to control how the system handles errors and what information is displayed when running commands.

Name	Description	Default
verbose	Enables more detailed command line output, including more information about errors	False
halt_on_error	Stop the execution of run commands when the system encounters a fatal rule error. When set to false the system will log fatal rule errors and output to a review file	False
validate_regex_rules	Validate regular expression rules when conversion command is run. When enabled all rules that are not validated will be added to a report file	True

[db]

The debug section allows a user to specify the connection attributes for connecting to the MySQL server.

Name	Description	Default
host	Specifies the host to be used when connecting to the project databases	localhost
Port	Specifies the port to be used when connecting to the project databases	

[db_stand]

The db_stand section allows a user to specify the database options for the standardized relational database.

Name	Description	Default
name	The name of the standardized relational database	[project_name]
collate	The collation for the standardized relational database	utf8_unicode_ci
encoding	The encoding for the standardized relational database	utf8

[db_recid]

The db_recid section allows a user to specify the database options for the recidivism relational database.

Name	Description	Default
name	The name of the recidivism relational database	[project_name]_recid
collate	The collation for the recidivism relational database	utf8_unicode_ci
encoding	The encoding for the recidivism relational database	utf8

[segment_subject]

For advanced users only, changing these settings could severely impact the conversion system.

Name	Description	Default
id	The unique identifier	1
label	The label of the segment file, this should match the name of the NLETS parsed file	subject
region_var	The variable that contains the region of the rap sheet record	rapstatex
priority	The load priority	1

[segment_arrest]

For advanced users only, changing these settings could severely impact the conversion system.

Name	Description	Default
id	The unique identifier	2
label	The label of the segment file, this should match the name of the NLETS parsed file	arrest
region_var	The variable that contains the region of the rap sheet record	rapstatex
priority	The load priority	2

[segment_sentence]

For advanced users only, changing these settings could severely impact the conversion system.

Name	Description	Default
id	The unique identifier	3
label	The label of the segment file, this should match the name of the NLETS parsed file	sentence
region_var	The variable that contains the region of the rap sheet record	rapstatex
priority	The load priority	3

[segment_supervision]

For advanced users only, changing these settings could severely impact the conversion system.

Name	Description	Default
id	The unique identifier	4
label	The label of the segment file, this should match the name of the NLETS parsed file	supervision
region_var	The variable that contains the region of the rap sheet record	rapstatex
priority	The load priority	4

[segment_demographic]

For advanced users only, changing these settings could severely impact the conversion system.

Name	Description	Default
id	The unique identifier	5
label	The label of the segment file, this should match the name of the NLETS parsed file	demographic
region_var	The variable that contains the region of the rap sheet record	rapstatex
priority	The load priority	5

3 - System Usage

3.1 Overview

Once the BJS CCHRRD Conversion system has been installed and a project has been created, project commands are now available to the user, whereas prior to new project creation only system commands could be issued without the potential for a system error. Listed below is an overview of all the system and project commands available to the user. At any time, a list of all available commands and their short descriptions can be seen using the help command-line parameter on the CCHRRD system command, as well as the short descriptions of each sub-command and parameter supported by the CCHRRD Conversion system.



```

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py -h

```



```

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py new_project -h

```

3.1.1 Command reference

System Commands

Name	Description	Type
Activate* ⁵	Set the current working project	Project Management
New Project	Create a new conversion project	Project Management
Add User	Add a user to the system	Account Management
Drop User	Remove a user from the system	Account Management
Update User	Update a user name or password	Account Management
Copy Rules*	Copy project business logic to system default business logic	Rule Management

⁵ *Requires at least one conversion project

Project Commands

Name	Description	Type
Stage	Load the NLETS intermediate data into the standardized relational database	Conversion
Pre Process	Do initial processing work on the standardized relational database	Conversion
Process	Finalize the standardized relational database and create SPSS extract files	Conversion
Transform	Transform the standardized relational database	Conversion
Load Recidivism Database	Load transformed standardized data into the recidivism relational database	Conversion
Load Recidivism Flat Files	Create the recidivism flat SPSS files	Conversion
Update Metadata	Update the project business logic metadata	Business Logic Management
Update Rules	Update the project business logic coding rules	Business Logic Management
Compare	Compare output files for differences	Quality Control
Validate Rules	Validate the project business logic coding rules	Quality Control
Cleanup	Remove files from the project folder	File Management

3.1.2 Conversion Life Cycle

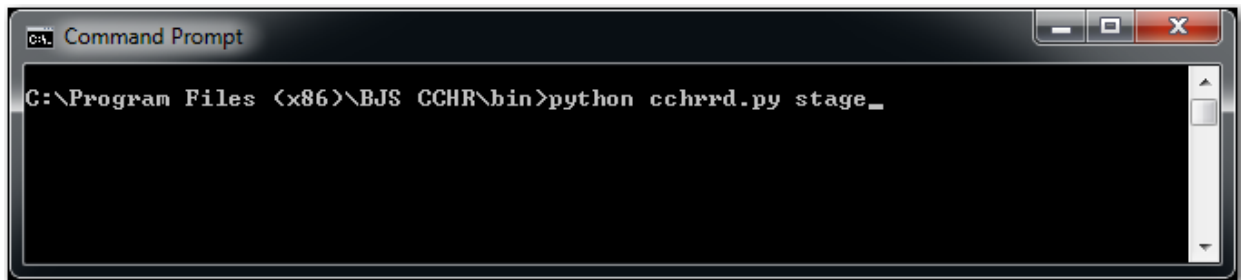
The first step after a new project is created is to place the NLETS intermediate parsed files into the input directory (*projectdir/rules*). The program requires a subject file, an arrest segment file, a sentence subject file, and a demographic segment file. The subject file should be in SPSS format (.sav) and the segment files should be in bar delimited format (.del). The subject file must contain one line for each subject in the study and must meet the specifications in the subject metadata file (*projectdir/metadata/subject/var_info.xls*). The arrest/sentence segment file must contain one arrest/court-sentencing record per line and should conform to the specifications in its metadata file (*projectdir/metadata/segment/var_info.xls*). If the files do not meet the specifications listed in the metadata files then either the metadata files must be updated or the subject/segment files need to be transformed to conform to the project metadata specifications. Please see section 4.1.3 for more information on reviewing and updating project metadata files.

After copying the input files into the input directory and verifying that these files adhere to the metadata file specifications, the conversion system is now ready to run conversion commands. The conversion cycle is a complex workflow and to make this workflow manageable the cycle has been broken into multiple phases. Below is a high level overview of each phase in the conversion process. This is followed by a detailed description of all available commands including a detailed description of the command,

information on the various parameters that can be specified, and the necessary quality control steps that need to take place before and after each command.

Data Staging

The first phase in the conversion cycle, **data staging**, creates the standardized and recidivism relational databases and loads the segment files into the standardized database. The command to execute this phase is the stage command.



```
Command Prompt
C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py stage_
```

Data Pre-Processing

After the databases have been created and the intermediate files are loaded the standardized database is prepped for the **pre-process** phase. During this phase the pre-process business logic coding rules are applied to the staged data.



```
Command Prompt
C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py pre_process
```

Data Processing

The next step is to apply the process business logic coding rules to the pre-processed data. After this step is complete the standardized relational database and SPSS file extracts will be available to the user.



Recidivism Database Creation

After the process phase is complete the next step is to create the recidivism relational database. Creating the recidivism relational database is a two-step process. The first step is to transform the data contained in the standardized relational database into a format suitable for loading into the recidivism database.



The second step is to take the transformed data and load it into the recidivism relational database.



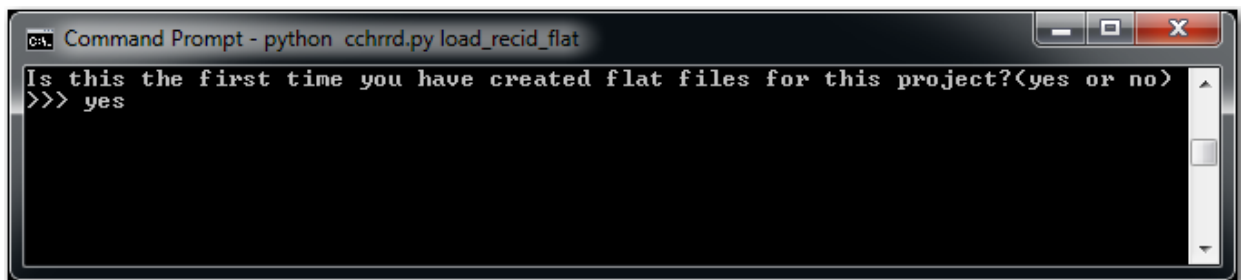
After the load_recid_db command completes, the recidivism relational databases and SPSS extract files will be available to the user.

Recidivism Flat File Creation

The final step is to create the Demographic Flat, Criminal History Flat, and Criminal History Summary SPSS flat files. This is accomplished by invoking the load_recid_flat command.



Invoking this command will prompt the user for input on what files they would like to load. For the first run of the conversion cycle a user should always respond “yes” to the first question:



After this command is finished running the demographic and summary SPSS flat files will be available for review. However, due to the size of the Criminal History Flat file there is an additional step of loading this file from a raw text format into SPSS. Please reference the below Load Recidivism Flat Files command documentation for more information on how to load the Criminal History Flat file into SPSS.

3.2 System Commands

3.2.1 Activate

Command Type: System (project management)

Summary

The BJS CCHRRD Conversion system allows a user to create and run multiple projects. When a project command is invoked it will only be applied to the current working project. Therefore, when working with multiple projects it is necessary to run the Activate command to switch between projects.

For example, consider that the conversion system has two projects *demo1* and *demo2*. If *demo1* is the current working project and the user wants to run conversion commands on *demo2* the user must invoke

the activate command to set *demo2* as the current working project and enable project commands to run on the *demo2* project.

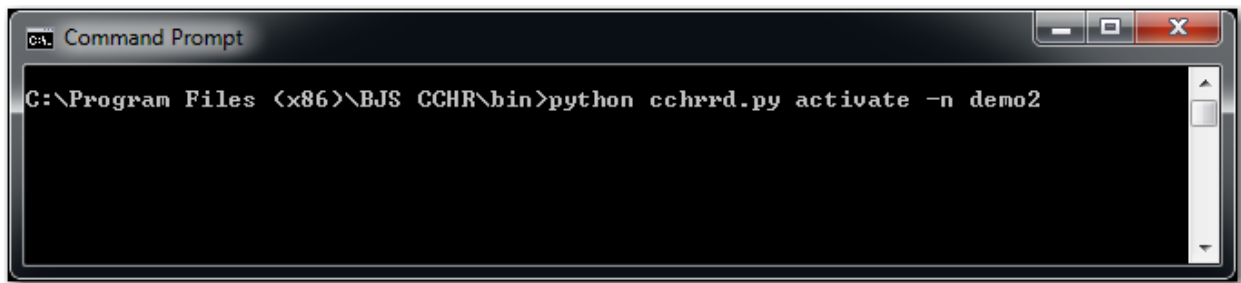
Syntax

```
activate -n [PROJECT NAME] (-r [ROOT DIR]) (-v)
```

Parameter	Type	Description	Default
--name, -n	Required	Specifies the name of the project to activate.	None
--root_dir, -r	Optional	Specifies the location of the root directory that contains project data.	[CONFIG.ROOT_DIR]
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

Usage Example

Example 1: Create a new project named demo2



3.2.2 New Project

Command Type: System (project management)

Prerequisites: None

Summary

Creates a new conversion project based on the parameters you provide. No parameters are necessary when creating a new project, all required information will be prompted for by the system.

When the new project command is invoked the system will prompt for all required information for project creation. After this information has been provided the system will create a new project directory structure in the program data directory, create the project configuration file, and set the new project to the current working project. The project's business logic will be based on the system default rules specified during project initiation or through the project copy rules command.

Input

1. Business logic

Output

1. Project folder containing directory structure, business logic, and project configuration file

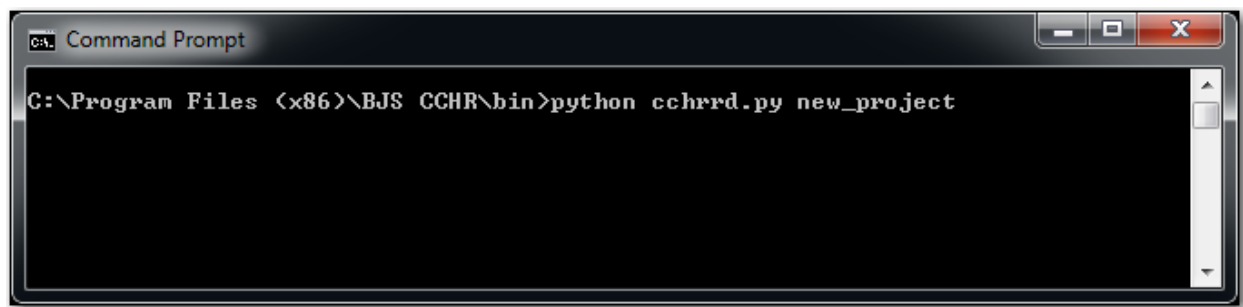
Syntax

```
new_project -n [PROJECT NAME]
```

Parameter	Type	Description	Default
--name, -n	Optional	Specifies the name of the new project.	None
--in_dir, -i	Optional	Specifies the path to the directory containing the NLETS intermediate parsed files.	[PROJECTDIR]\input
--metadata_dir, -m	Optional	Specifies the path to the directory containing the business logic metadata to load as the base metadata for the project.	[ROOTDIR]\build\metad ata
--rule_dir, -r	Optional	Specifies the path to the directory containing the business coding rules to use as the base coding rules for the project.	[ROOTDIR]\build\rules
--follow_period, -f	Optional	Specifies the number of years following a given prisoner's day of release for which the software will calculate recidivism statistics.	5
--host	Optional	Specifies the host to be used when connecting to the database.	[CONFIG.DB_HOST]
--port, -p	Optional	Specifies the TCP port of the MySQL server.	[CONFIG.DB_PORT]
--db_stand_name	Optional	Specifies the name to be given to the standardized database.	[name]
--db_recid_name	Optional	Specifies the name to be given to the recidivism database.	[name]_recid
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

Usage Examples

Example 1: Create a new project with system defaults



Example 2: Create a new project specifying the location where the business logic coding rules should be loaded from

3.2.3 Add User

Command Type: System (account management)

Summary

Add a user account to the system. When the user invokes the Add User command they will be prompted for the new user name and new user password.

Syntax

`add_user (-v)`

Parameter	Type	Description	Default
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

Usage Example

3.2.4 Drop User

Command Type: System (account management)

Summary

Remove a user account from the system. The user will be prompted to enter the username and confirm that the username before removing the user from the system. After the user is removed from the system they will no longer be able to execute system or project commands.

Syntax

`drop_user (-v)`

Parameter	Type	Description	Default
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

Usage Example

```

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py drop_user

```

```

Please enter user to drop: demo_user
Please confirm the user to drop: demo_user
You are about to drop the following user from the system: demo_user
are you sure you want to process (y/n)?
>>> y

```

3.2.5 Update User

Command Type: System (account management)

Summary

Update a user account credentials. After invoking the command the user will be able to select from updating a user's name or password. The system will prompt for new name and/or password following system account protocol.

Syntax

update_user

Parameter	Type	Description	Default
--verbose, -v	Toggle	Enables more detailed output, including more information about errors	Off

Usage Examples

```

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py update_user

```

```

Please select from the following options:
1: Update username
2: Update user password
3: exit
>>>

```

3.2.6 Copy Rules**Command Type:** System (rule management)**Summary**

Copies project business logic to system default business logic. This command will prompt the user to select a project and then prompt for confirmation before copying business metadata and business coding rules.

During software initialization the user selects the system default business logic based on the business logic available during software installation. When a new project is created this default business logic is loaded as the project business logic. New rules and updates to existing rules may happen while running this new project. The copy_rules command allows users to copy the business logic from a new project to the system default business logic. After running copy_rules, all new projects will load the updated system default business logic as their base business logic.

Syntax`copy_rules (-v)`

Parameter	Type	Description	Default
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

Usage Examples**Example 1:** Run the Copy Rules command

```

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py copy_rules

```

```

Please select a project to use as the new default business logic
1: demo
2: demo1
0: Exit
>>> 1

You are about to copy
d:\projects\project_data\demo\metadata
TO
d:\projects\build\metadata

Do you wish to proceed (yes or no)?
>>> yes

```

3.3 Project Commands**3.3.1 Stage****Command Type:** Project (conversion)**Prerequisites**

NLETS intermediate files must be present in the input directory and adhere to segment metadata specifications.

Summary

The stage command starts the first phase in the conversion cycle, **data staging**, which creates the standardized and recidivism relational databases and loads the segment files into the standardized database.

Once the user invokes the command the system will create a series of tables in the two databases based on the business logic metadata. The system will create the archive tables for each segment and all the lookup tables. Please see section 4.1.2 for more information on how to update the relational databases using metadata files.

After the standardized and recidivism databases are created the system loads the NLETS intermediate data files, located in the input directory, into the standardized relational database. For each segment the bar delimited files will be saved into the segment archive table. After the data is loaded into the archive tables the system will update the length of the original variables to the shortest length possible (each variables length is set to the longest string present in the data that corresponds to each variable).

Input

- 1) NLETS intermediate parsed files
- 2) Project business logic database and segment metadata

Output

- 1) MySQL standardized relational database
- 2) MySQL recidivism relational database
- 3) Segment standardized database archive tables

Syntax

stage

Parameter	Type	Description	Default
--no_db	Toggle	Do not rebuild database, simply reload the NLETS intermediate input files	False
--segments, -s	Optional	Specifies the segments to load into the standardized relational database. Uses a bar-delimited list; for example, "su a c" would run the stage command for the subject, arrest and court segments.	su a c s d
--verbose, -v	Toggle	Enables more detailed output, including more information about errors	Off

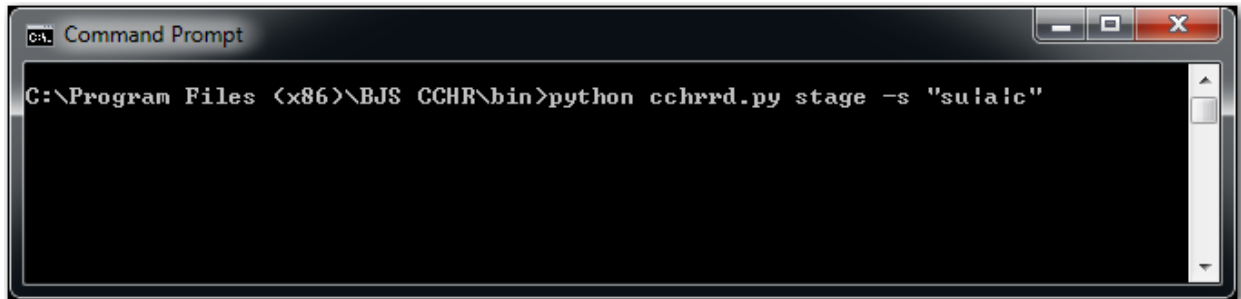
Usage Examples

Example 1: Run the stage command building both databases and load all segment files into the standardized relational database.



```
C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py stage_
```

Example 2: Run the stage command building the databases and load the subject, arrest, and court-sentencing segment files into the standardized relational database.



```
C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py stage -s "su!a!c"
```

3.3.2 Pre-Process

Command Type: Project (conversion)

Prerequisites

Stage phase has completed successfully i.e. the databases were created and the NLETS intermediate data was loaded into the standardized database archive tables.

Summary

The `pre_process` command starts the second phase in the conversion cycle, **data pre-process**. This phase applies the pre-process business logic coding rules to the archive tables in the standardized relational database. The pre-process phase will run all the coding rules present in the pre-process rule steps in *projectdir/rules*, where each step corresponds to a folder that starts with “pre_process_rules” and ends with a number indicating the order of the step. Please see section 4.3 for more information on the pre-

process business logic, including current state of rules and how to update these rules when running a new cohort.

As the pre-process rules are applied all rule output will be saved to the log directory and these logs will be transformed into reports detailing all warning and errors that occurred during pre-processing. The pre-process rules will create and populate new columns in the archive tables that will be used during data processing.

Input

2. Pre-process business logic coding rules
3. Standardized relational database segment archive tables

Output

- 1) Updates the standardized relational database segment archive tables
- 2) Pre-process conversion reports are saved to *projectdir/output/report/pre-process/segment/all*

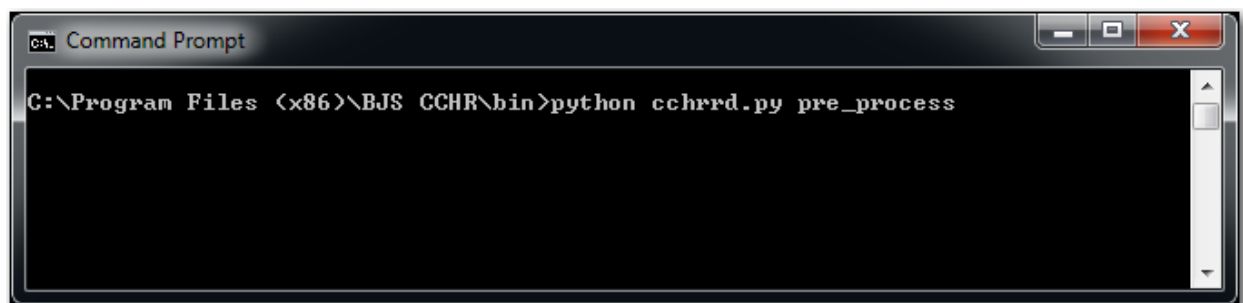
Syntax

```
pre_process (-s (a|c|s|d)) (-v)
```

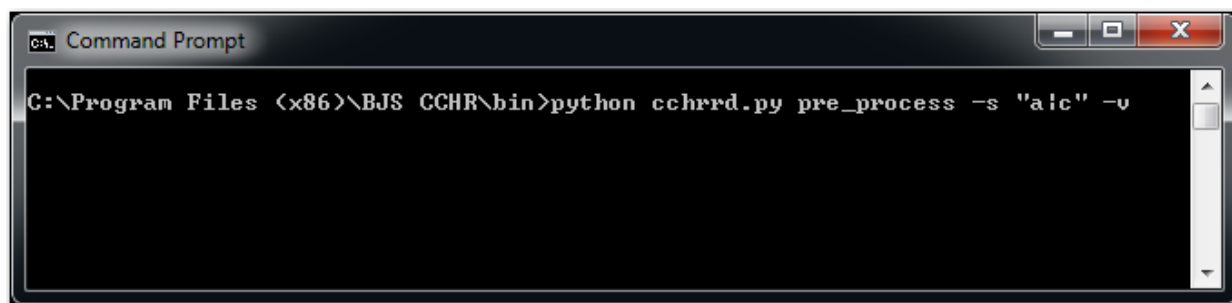
Parameter	Type	Description	Default
--segments, -s	Optional	Specifies the segments to pre-process. Uses a bar-delimited list; for example, "a c" would run data preprocessing for the arrest and court segments.	a c s d
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

Usage Examples

Example 1: Run pre-process command on all segments.



Example 2: Run pre-process command on arrest and sentence segment with verbose output turned on.



3.3.3 Process

Command Type: Project (conversion)

Prerequisites

The stage and pre-process phases are complete. i.e. the databases were created, the NLETS intermediate data was loaded into the standardized database archive tables, and these tables had the pre-process coding rules applied to create new columns/column values.

Summary

The process command starts the third phase in the conversion cycle, **data processing**. This phase applies the process business logic coding rules to the archive tables in the standardized relational database. The pre-process phase will run all the coding rules present in the process rule steps in *projectdir/rules*, where each step corresponds to a folder that starts with “process_rules” and ends with a number indicating the order of the step. Please see section 4.4 for more information on the process business logic, including current state of rules and how to update these rules when running a new cohort.

As the process rules are applied all rule output, including conversion errors, will be saved to the log directory and these logs will be transformed into reports detailing all warning and errors that occurred during the process phase. After the process rules are applied the processed data will be saved to the segment tables in the standardized relational database. On the initial run of the process command the segment tables will be created based on the business logic metadata. Additionally, an SPSS file is created containing the data present in the standardized database segment tables.

The SPSS file and conversion reports provide the set of information that a quality control team will use to ensure that the standardization process has completed successfully. If the standardized segment tables do not pass the quality control review, the process and/or pre-process business logic coding rules will need to

be updated and these files will need to be rerun. To help with this quality control cycle the process command supports a number of parameters with the most commonly used being the region parameter (--region, -r), which allows the process phase to be run on only the records where the value in its region_var (see configuration file) equals the supplied region parameter. For example, the process phase is run and the conversion output report shows that the Arizona region had conversion error. The user would then update the Arizona business logic rule files and then rerun the process phase providing the -r parameter to only load the Arizona records for data processing.

After the segment files have all been reviewed and no errors detected, the standardized relational database is complete and the recidivism phase is ready to transform and then load the standardized data into the recidivism outputs.

Input

1. Process business logic coding rules and metadata
2. Standardized relational database segment archive tables

Output

1. Updates the standardized relational database segment tables
2. Segment SPSS database extract and process conversion reports
 - a. *projectdir/output/review/set/segment/region/round*

Syntax

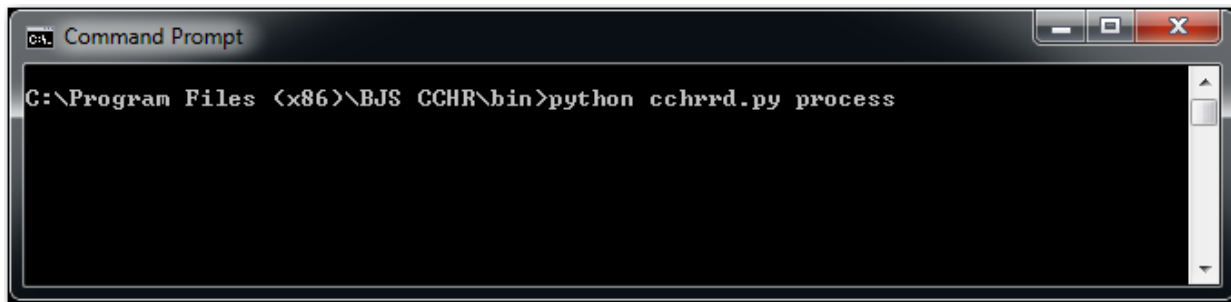
process (-s (segment list)) (-o “format”) (-l int) (-w “where string”) (-r (region list)) (-i “rule name”) (-e “rule name”) (-p) (-v)

Parameter	Type	Description	Default
--segments, -s	Optional	Specifies the segments to process. Uses a bar-delimited list; for example, “a c” would run this command for the arrest and court segments. For a summary comparison, use “summary”.	a c s d
--out_format, -o	Optional	Specifies the file format for saving the processed files.	spss
--limit, -l	Optional	Sets the number of records to load	None
--where, -w	Optional	SQL where clause applied on data loading.	None
--regions, -r	Optional	Specifies a region to load.	all
--include_rules	Optional	Pass this parameter one or multiple times to only include a specific rule manager(s) for process phase. Example: -i ancic -i acnt	None
--exclude_rules	Optional	Pass this parameter one or multiple times to exclude specific rule manager(s) from process phase. example: -i cncic -i ccnt	None

Parameter	Type	Description	Default
--parallel, -p	Toggle	Enables parallel processing.	False
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

Usage Examples

Example 1: Run process command on all segments (this command should be used on initial run.)

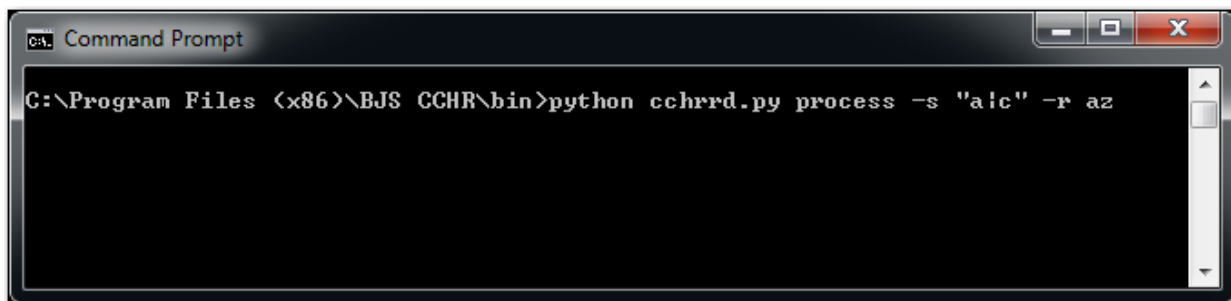


```

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py process

```

Example 2: Run process command on the arrest and court-sentencing segments for all records that are from the Arizona region



```

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py process -s "a|c" -r az

```

3.3.4 Transform

Command Type: Project (conversion)

Prerequisites

The recidivism relational database has been created and the standardized relational database has been created, loaded, processed and reviewed (stage, pre-process, and process phases are complete).

Summary

Transform the standardized relational database into recidivism format by applying the recidivism business logic. The transform command supports a number of parameters to control the operation of the transform phase and application of recidivism rules. For controlling the recidivism rules, the transform command supports:

- 1) a toggle for including or removing juvenile records from the recidivism database and flat files
- 2) an offense deletion flag which will remove all records where the record's charge code value matches the value passed to the parameter
- 3) a follow-up period flag to set the number of years that recidivism statistics should be tracked after the subjects date of release (this parameter override the project configuration follow_period setting)

To increase the performance of the transform phase the number of cores and threads can be increased in the project configuration file or by using the transform command parameters (see below). The default setting is 2 cores and 100 transform threads. Increasing these settings may result in system errors by consuming more resources than the system have available. If the system you are using has 4 high performance processors and 16GB of memory the cores can be increased to 4 and the threads can be increased. The amount the threads can be increased is dependent on the highest number of arrest cycles for the subjects in a particular cohort. If the number of arrest cycles is between 50 and 100, the transform threads can be increased to a maximum of 175, but if the highest number of arrest cycles is greater than 100 the transform threads setting should not be increased.

Input

- 1) MySQL standardized relational database segment tables
- 2) Project business logic recidivism rules

Output

- 1) Creates temporary files which are used by the Load Recidivism Database command

Syntax

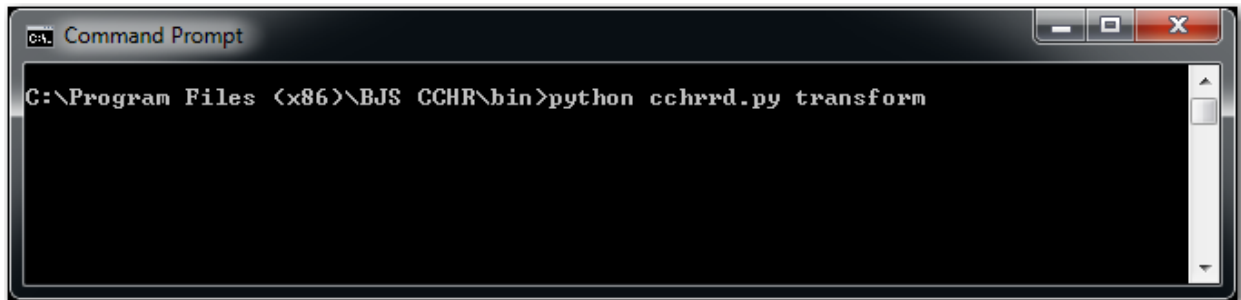
transform

Parameter	Type	Description	Default
--cores, -c	Optional	Specifies the number of cores to use, which determines how many sub processes are spawned.	[CONFIG.CORES]
--thread, -t	Optional	Specifies the number of threads to use on each sub process.	[CONFIG.TRANSFORM_THREADS]

Parameter	Type	Description	Default
--juv_flag, -j	Toggle	Pass this parameter to turn on flagging of juvenile records	False
--offense_deletion_flags, -o	Optional	Pass this parameter zero or more times to flag offenses for removal from the recidivism database; by default traffic offenses (550) are flagged for removal	550
--follow_period, -f	Optional	Specifies the number of years following a given prisoner's date of release for which the software will calculate recidivism statistics.	[CONFIG.FOLLOW_PERIOD]
--verbose, -v	Toggle	Enables more detailed output, including more information about errors	Off

Usage Examples

Example 1: Run the transform phase with default settings



```

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py transform

```

Example 2: Run the transform phase setting the number of processors to 4 and the number of threads to 150

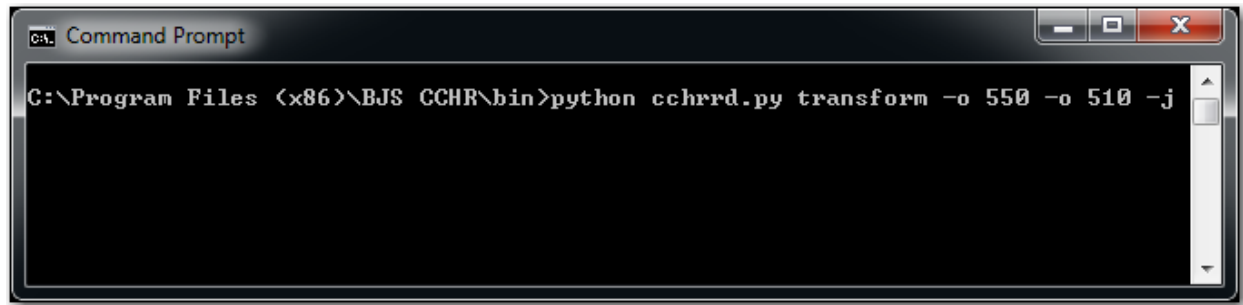


```

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py transform -c 4 -t 150

```

Example 3: Run the transform phase removing traffic, riot offense, and juvenile offenses



3.3.5 Load Recidivism Database

Command Type: Project (conversion)

Prerequisites

The recidivism relational database has been created and the standardized relational database has been created, loaded, processed and reviewed (stage, pre-process, and process phases are complete).

Additionally, the transform command has been run creating the recidivism temporary files.

Summary

The Load Recidivism Database command is the second step in the recidivism database creation. After the recidivism temporary files have been created during the previous Transform step the Load Recidivism Database command loads the temporary files into the MySQL recidivism relational database. The data from the temporary files are loaded into two tables, the arrest and sentence segment tables. After the data is loaded four SPSS files are created: arrest, sentence, arrest_del_records, and sentence_del_records. The first two files are a flat file extract of the arrest and sentence segment tables in the MySQL recidivism database. The second two files provide are all the arrest and sentence records that were removed from the standardized database before loading into the recidivism database due to recidivism coding rules. For example, if traffic, riot, and juvenile offenses were all flagged for removal all these cases would be present in the del_records files.

Input

- 1) Recidivism temporary files
- 2) Business logic metadata for the recidivism database segment tables
- 3) MySQL recidivism relational database

Output

- 1) MySQL recidivism relational database segment tables
- 2) Recidivism database segment table SPSS flat file extracts and SPSS deleted records files
 - a. Files are saved to *projectdir/output/transform*

Syntax

```
load_recid_db (-v)
```

Parameter	Type	Description	Default
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

Usage Examples

Example 1: Load the recidivism relational database



3.3.6 Load Recidivism Flat Files

Command Type: Project (conversion)

Prerequisites

The standardized and recidivism relational databases are loaded (stage, pre-process, process, recidivism transformation and database load phases all complete).

Summary

The final step in the conversion cycle is to create the recidivism SPSS flat files. The flat files are created by extracting data from the recidivism relational database, applying a series of transformations to the data, and then loading the transformed data into the SPSS flat files. The output command supports several parameters to control the creation of the recidivism SPSS flat files. The cores and thread parameters support the efficiency of flat file creation (recommended to not change these settings). The follow_period parameter will filter records from the recidivism database that are outside the maximum number of years

after the date of release in the subject file. Since the recidivism database was already created with a follow-up period this parameter can only further decrease the window of the follow-up period for flat file creation, it cannot obtain records that are outside the follow-up period of the recidivism database. This is useful if the researcher would like to have multiple SPSS recidivism flat files with different follow-up periods.

Upon invoking the command the user will be prompted to tell the system if it is the first time running the program:

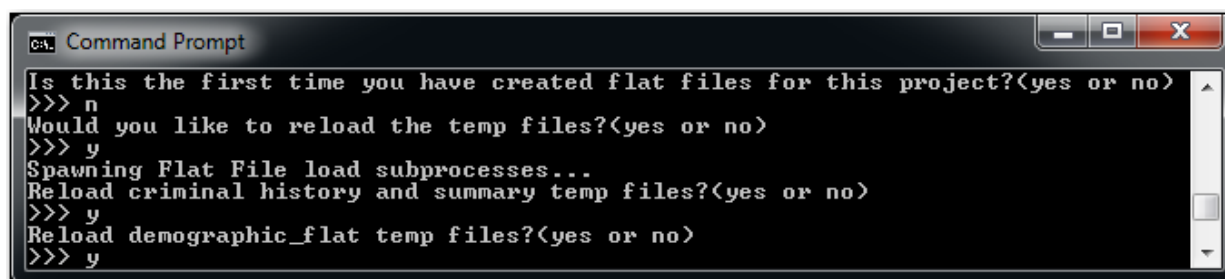


```

C:\> Command Prompt - python cchrrd.py load_recid_flat
>>>
Is this the first time you have created flat files for this project?<yes or no>
>>>

```

If the user selects yes all steps will be carried out and the user will not need to provide any additional information. The system will spawn up a number of transformation threads and create temporary files to speed-up flat file creation. If the user responds “no”, then the system will ask the user if they would like to reload the temporary files. If the user responds no then the old temporary files will be used, if the user responds with “no” then the system will prompt the user if they would like to reload the demographic temporary files, and the criminal history and summary temporary files. The temporary files need to be reloaded if a new follow-up period is specified or if any of the recidivism transformation rules are updated.

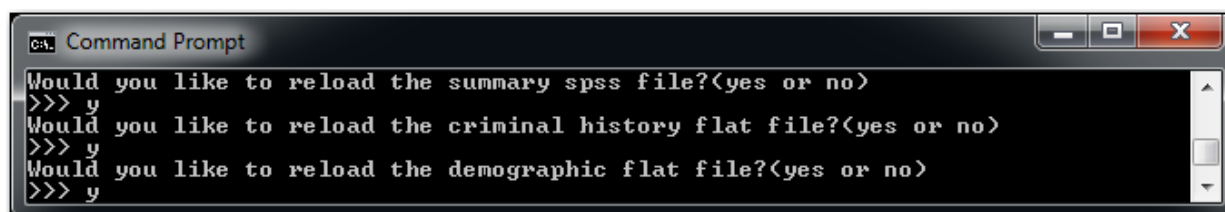


```

C:\> Command Prompt
Is this the first time you have created flat files for this project?<yes or no>
>>> n
Would you like to reload the temp files?<yes or no>
>>> y
Spawning Flat File load subprocesses...
Reload criminal history and summary temp files?<yes or no>
>>> y
Reload demographic_flat temp files?<yes or no>
>>> y

```

The system will then prompt the user if they would like to reload the flat files.



```

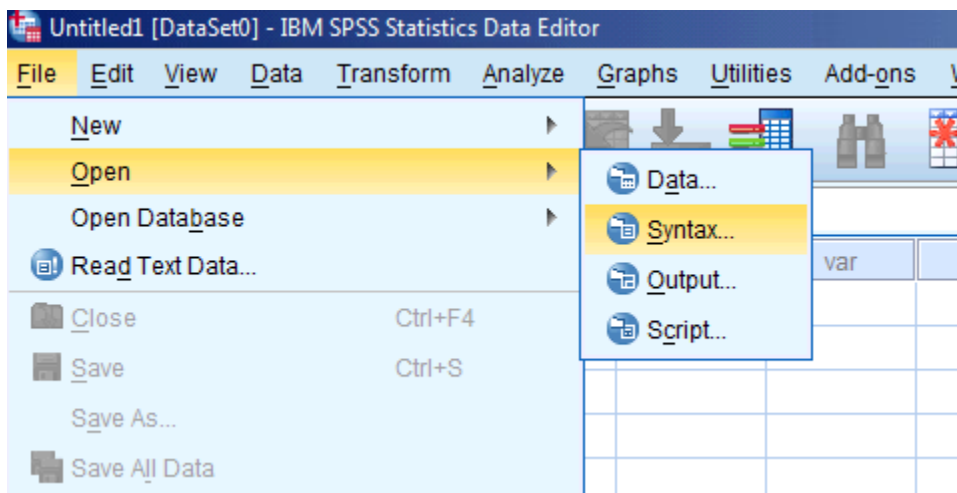
C:\> Command Prompt
Would you like to reload the summary spss file?<yes or no>
>>> y
Would you like to reload the criminal history flat file?<yes or no>
>>> y
Would you like to reload the demographic flat file?<yes or no>
>>> y

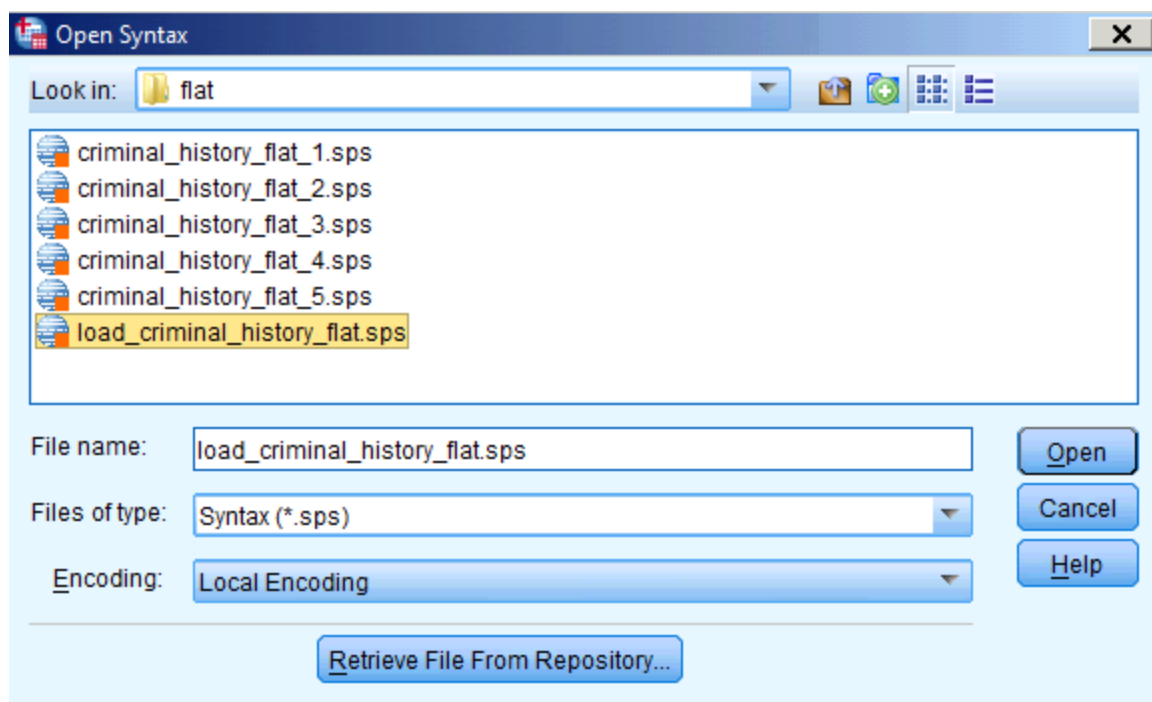
```

After the Load Recidivism Flat files command is finished the demographic and summary SPSS output files will be located in their respective directories within the *projectdir/output/recid* folder. However, the Criminal History Flat file will not be loaded into SPSS. The user will need to load a text file into the SPSS format using the SPSS program and sps syntax files. These steps are required due to the large number of columns and size of the Criminal History Flat file.

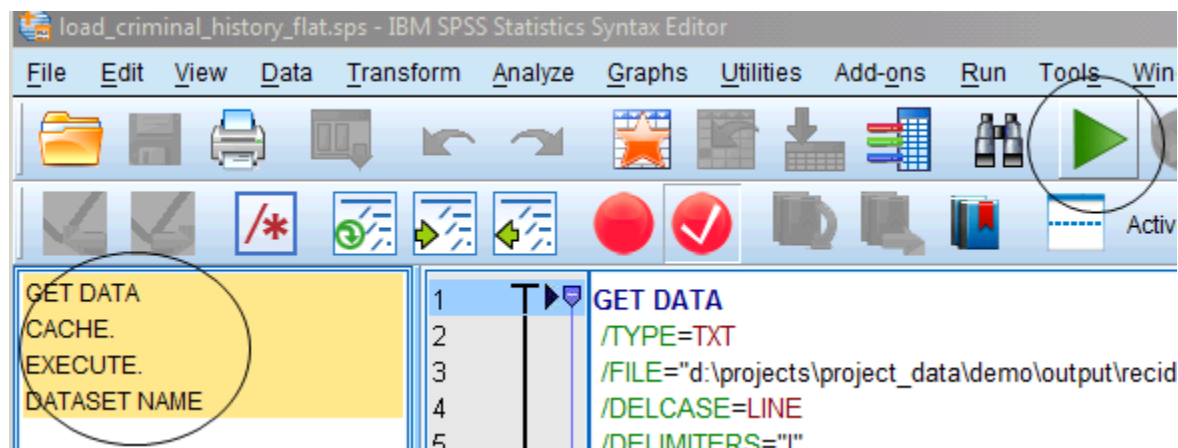
To load the Criminal History Flat file from its outputted text format preform the following steps:

- 1) Open SPSS
- 2) Open the load_criminal_history_flat sps file located in the *projectdir/output/recid/flat* directory





- 3) After the load_criminal_history_flat file is loaded in run all the sps syntax commands by selecting them on the left on and pressing the green execute button.



- 4) After the data has finished loading into SPSS, repeat steps two and three for each of the criminal_history_flat_id sps files in the *projectdir/output/recid/flat* directory to add value labels to the loaded data.
- 5) Finally save the SPSS file to your desired location

Input

- 3) MySQL recidivism relational database
- 4) Business logic metadata for the recidivism flat files

Output

- 2) Criminal History Flat SPS syntax value label and data load syntax files
 - a. Located at *projectdir/output/recid/flat*
- 3) Criminal History Demographic SPSS flat file
 - a. Located at *projectdir/output/recid/demographic/demographic_flat.sav*
- 4) Criminal History Summary SPSS flat file
 - a. Located at *projectdir/output/recid/summary/criminal_history_summary.sav*

Syntax

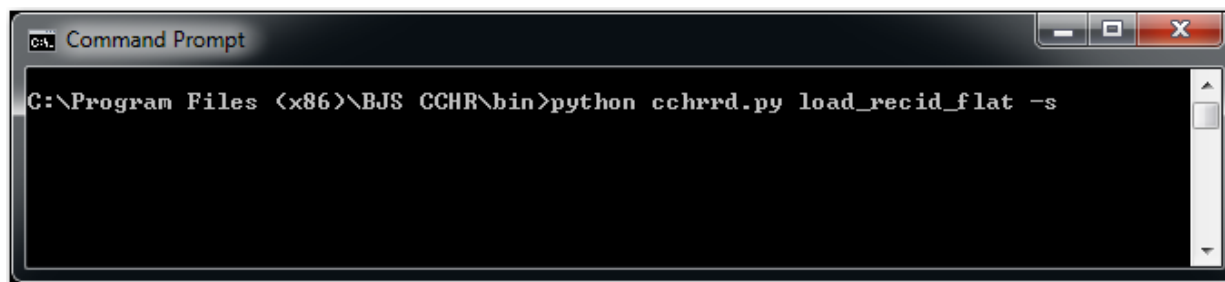
load_recid_flat

Parameter	Type	Description	Default
--cores, -c	Optional	Specifies the number of cores to use, which determines how many sub processes are spawned.	[CONFIG.CORES]
--thread, -t	Optional	Specifies the number of threads to use on each sub process.	[CONFIG.TRANSFORM_THREADS]
--follow_period, -f	Optional	Specifies the number of years following a given prisoner's day of release for which the software will calculate recidivism statistics.	[CONFIG.FOLLOW_PERIOD]
--sort, -s	Toggle	The flat files are sorted by casenum	On
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

Usage Examples

Example 1: Create the recidivism research flat files using system defaults



Example 2: Create the recidivism research flat files sorted by casenum or id


```

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py load_recid_flat -s

```

3.3.7 Update Metadata

Update the current project's business logic metadata. The update metadata command will transform the project's segment, database, and flat file metadata from the excel input format into json format. The json files are used by the conversion system so it is necessary to run the update metadata command to load in any changes made by the user to the excel metadata input files. Please see section 4.1 for more information on the project business logic metadata.



```

C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py update_metadata

```

Command Type: Project (business logic management)

Syntax

update_metadata

Parameter	Type	Description	Default
--verbose, -v	Toggle	Enables more detailed output, including more information about errors	Off

3.3.8 Update Rules

Update the current project's business logic coding rules. The update rules command will transform the project's pre-process and process coding rules from the Excel crosswalk format into the ruleset text format. The ruleset files are used by the conversion system so it is necessary to run the update rules

command to load in any changes made by the user to the excel coding rules input files. Please see section 4.2 for more information on the project business logic coding rules.

When the update rules command is invoked the system will prompt the user for which type of rule(s) to update

Command Type: Project (business logic management)

Syntax

update_rules

Parameter	Type	Description	Default
--region, -r	Optional	The region to load; pass this argument multiple times to load multiple regions. Example: "-r az -r mn".	all
--all_regions	Toggle	Load all the regions.	Off
--no_replace		Add this parameter if you are sure that all your key columns are text. This will not replace the .0 excel adds to integer values on the key columns.	Off
--wvfb	Toggle	Toggle to load WVFBI global rules.	Off
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

3.3.9 Compare

Command Type: Project (quality control)

Prerequisites

The compare command requires at least two rounds to be present in the *projectdir/output/review/segment/region* directory for process comparisons and at least two rounds in the *projectdir/output/review/summary* directory.

Summary

Compare output files to detect differences between current and previous round. This function is used as a quality control measure to ensure that changes made to the business logic coding and recidivism rules produced the desired results. Currently allows comparisons for process output and Criminal History Summary (summary) file output. When the compare command is invoked it will use the default settings which are to compare the full arrest and sentence datasets.

This command will produce a difference report "report_diff.xls" to the latest round in the review directory if differences are detected from the prior process round. If no differences are detected the difference report will not be created. See section 5.4 for more details on the comparison reports.

Input

- 4) Two rounds of SPSS database extract file for arrest or court-sentencing segment
or
- 5) Two rounds of SPSS Criminal History Summary File

Output

- 3) Difference comparison report
 - a. Process comparisons: *projectdir/output/review/segment/region/highest_round*
 - b. Summary Comparison: *projectdir/output/review/summary/highest_round*

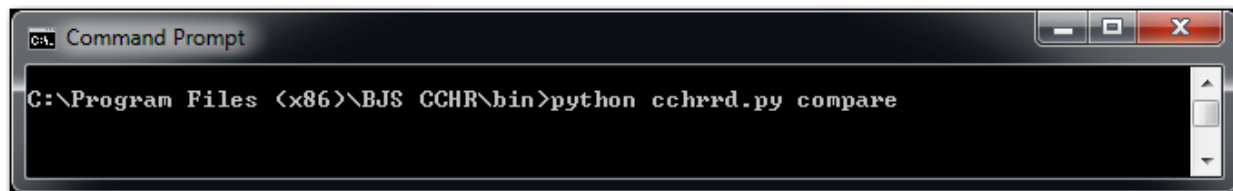
Syntax

compare

Parameter	Type	Description	Default
--segments, -s	Optional	Specifies the segments to compare. Uses a bar-delimited list; for example, "a c" would run this command for the arrest and court segments. For a summary comparison, use "summary".	a c
--region, -r	Optional	Specifies the name of the region to compare. Use "all" to run this command for all regions.	all
--set	Optional	Specifies the name of the set to be used in the comparison.	[CONFIG.CURRENT_SET]
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

Usage Examples

Example 1: General compare command, equivalent to passing the "a|c" to the segment parameter and "all" to the region parameter.



Example 2: Same as example 1 but using parameters instead of system defaults

```
C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py compare -s "a|c" -r "all"
```

Example 3: Run a comparison of the summary file by passing “summary” to the segment parameter

```
C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py compare -s "summary"
```

Example 4: Run a comparison of the Arizona region for the arrest segment by passing “a” to the segment parameter and “az” to the region parameter

```
C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py compare -s a -r az
```

3.3.10 Validate Rules

Validate a project’s business logic pre-process and process coding rules. When the `validate_rules` command is invoked the system will load the business logic coding rules and produce reports listing out any warnings and errors detected within the coding rules. The `validate rules` command will check for referential integrity within crosswalks and that all regular expression rules are valid.

```
C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py validate_rules
```

The system will output all rule validation reports to *projectdir/output/report/validate_rules*. There are two possible reports that be saved to the above directory; a referential integrity report (report_ref_integ_error.xls) and a regular expression validation report (report_regex_init.xls). See section 5.5 and 5.6 for a detailed description on what is created in these reports and how to use this information to ensure the validity of conversion rules.

Command Type: Project (quality control)

Syntax

validate_rules

Parameter	Type	Description	Default
--meta_dir, -m	Optional	Specifies the directory containing segment metadata files.	[CONFIG.META_DIR]
--rules, -r	Optional	Specifies the root directory of the processing rules	[CONFIG.PROCESS_RULES_DIR]
--verbose, -v	Toggle	Enables more detailed output, including more information about errors	Off

3.3.11 Cleanup

Command Type: Project (file management)

Summary

This command will remove temporary project files. This command is typically run at the conclusion of a project to free up space. Running this command will remove temporary files that contain debug information, the recidivism temporary files that are needed for loading the recidivism flat files, and the process output files. After running cleanup the user will need to rerun the Transform and Load Recidivism Flat files command to recreate the recidivism flat files. After invoking the cleanup command the user will be prompted to confirm deletion of temporary files.

Syntax

cleanup (-v)

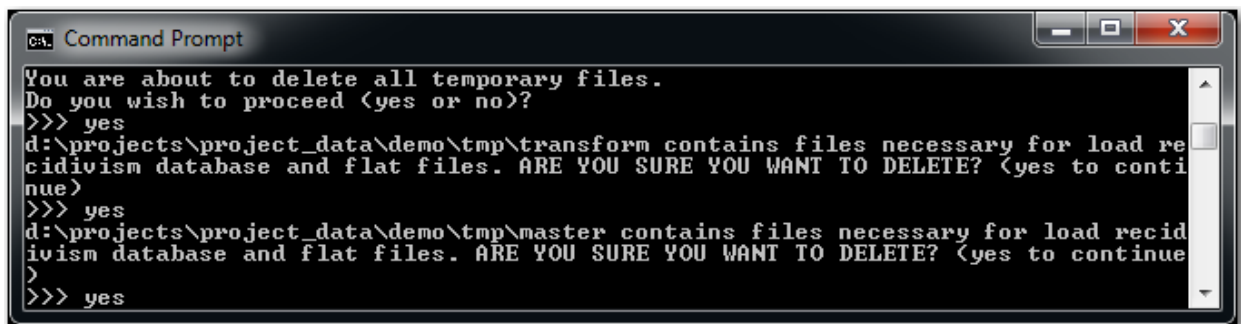
Parameter	Type	Description	Default
--verbose, -v	Toggle	Enables more detailed output, including more information about errors.	Off

Usage Example

Example 1: Run cleanup command to remove temporary files



```
Command Prompt
C:\Program Files (x86)\BJS CCHR\bin>python cchrrd.py cleanup
```



```
Command Prompt
You are about to delete all temporary files.
Do you wish to proceed (yes or no)?
>>> yes
d:\projects\project_data\demo\tmp\transform contains files necessary for load re
cidivism database and flat files. ARE YOU SURE YOU WANT TO DELETE? (yes to conti
nue)
>>> yes
d:\projects\project_data\demo\tmp\master contains files necessary for load recid
ivism database and flat files. ARE YOU SURE YOU WANT TO DELETE? (yes to continue
)
>>> yes
```

4 - Business Logic

4.1 Business Logic Metadata

4.1.1 Overview

The BJS CCHRRD Conversion system extracts data from parsed criminal history records, transforms this data into a format suitable for recidivism research and finally loads this transformed data into two relational databases and several flat files. The conversion software provides the platform to conduct this complex extract, transform, load (ETL) process. The business logic acts as the brains of the operation; it provides the system with information and rules on how to perform the ETL cycle. As of writing this manual, three cohorts have been processed through the full ETL cycle. Given that the NLETS parsing program changes over time and the desired transformation logic may differ for each cohort, each project has its own set of business logic. When the conversion system is installed the user must decide which previous project should be used as the default business logic. After making this decision all subsequent new projects will use the default business logic.

The business logic consists of two main components; the **business logic metadata** provides information on the database and flat file specifications and the **business logic coding rules** provide detailed instructions on how to carry out conversion transformations.

4.1.2 Database Metadata

During the first phase of the conversion life cycle, data staging, the standardized and relational database are created as are all of the lookup tables within these two databases. To achieve efficiency and adhere to database normalization practices the lookup tables, or relation tables, provide an indexed foreign key value to be used in the segment and segment archive tables.

The lookup tables are created based on the lookup table file specifications provided in the database metadata directory: *projectdir/metadata/database/standardized/lookups* and *projectdir/metadata/database/recidivism/lookups*. These two folders contain a series of Excel files. The Excel files provide an easy-to-use interface for users to control the creation of the database lookup tables without having to know any SQL. For each of the Excel files a table will be created in either the standardized or recidivism database with the table name matching the Excel filename, the table columns matching the Excel column headers, and the table rows matching the Excel row data. Each of these files must contain an **id** column that has the value “id” in the column header and a unique integer value for each subsequent row value. This column acts as the primary key field for the lookup table. The Excel file

must also contain a column named **label** with each subsequent row containing the description or label of the id value. A number of additional columns can be added, each of the columns will use the Excel file column header for the MySQL table column name and all of the row data will be stored in string format.

For example the ncic.xls file contains three columns; id, label, and type. The id column header identifies the primary key values for the table. The label contains the description/label associated with these values and the type stores what type of offense the code describes. The id column also serves as a foreign key column for the ancic column in the arrest and arrest archive tables and the cncic column in the court-sentencing and court-sentencing archive tables.

id	label	type
101	Treason	SOVEREIGNTY
102	Treason Misprison	SOVEREIGNTY
103	Espionage	SOVEREIGNTY
...	...	...

After the stage phase is complete all updates to the lookup tables need to be either completed using SQL queries or the conversion cycle needs to start over at the stage phase.

4.1.3 Segment Metadata

The MySQL standardized database contains a table for each of the segment files (subject, arrest, court-sentencing, supervision, and demographic) and the recidivism database contains a table for the transformed arrest and court-sentencing segment data. In an effort to reduce the amount of technical experience required for system users Microsoft Excel files are provided as an interface to control the database segment table specifications. The following Excel metadata files are available for modification:

- 1) Subject segment table specifications for standardized and recidivism databases
projectdir/metadata/subject/var_info.xls
projectdir/metadata/subject/recid_var_info.xls
- 2) Arrest segment table specifications for standardized and recidivism databases
projectdir/metadata/arrest/var_info.xls
projectdir/metadata/arrest/recid_var_info.xls
- 3) Court-sentencing segment table specifications for standardized and recidivism databases
projectdir/metadata/sentence/var_info.xls
projectdir/metadata/sentence/recid_var_info.xls
- 4) Demographic segment table specifications for standardized database

projectdir/metadata/demographic/var_info.xls

- 5) Supervision segment table specifications for standardized database

projectdir/metadata/supervision/var_info.xls

Each of the Excel table metadata files contains a column header and then one row representing column metadata. The Excel files that control the specifications for the standardized tables (var_info) contain the following columns:

Header Name	Description
Index	The index specifies the column position
Name	The name of the MySQL column
Original	Indicates that the MySQL column is mapped to the archive table. Additionally, this tells the system the position of the referenced field in the NLETS parsed
Format	The MySQL column format
Length	The MySQL column length
Precision	The precision of the MySQL column (if format=float)
FK	The table that the MySQL column is linked to as a Foreign Key
Label	The SPSS database extract file variable label
Value Labels	The SPSS database extract file variable value labels

The **index** column is a required field and should always be a unique integer value ranging from 0 to n, where n equals the number of rows present in the metadata file. The **index** is used to set the order of the columns when creating the standardized segment tables.

The **original** value should only be set on variables that are present in the NLETS parsed file. When the archive tables are created they will only include columns where the **original** field has an integer value. The values for the original field must range from 0 to n, where n equals the total number of fields in the NLETS bar delimited segment file. The original field not only indicates a column for the archive table, but it also informs the system of the position of the variable within the NLETS segment field. For example, on row 4 of the arrest var_info.xls file the row has a **name** value = casenumx and an **original** value = 0. This tells the system that the casenumx variable is the first field in the NLETS file and that this column should be added to the arrest segment archive table.

The **format** field is required for all rows and should be set to “string”, “integer”, “boolean”, or “float”.

The **length** field is required for all rows. The user should update the length field for only variables that are not present in the NLETS segment files (do not have an original value in the metadata file). The system

will update all the original values based on the longest detected string for each of the variables. This sets the length to the smallest size possible while eliminating the risk of data truncation.

The **precision** field is required for all rows that have a format equal to “float” and should be set to the desired number of decimals to be displayed after the decimal. For example, the precision should be set to 2 if the column should contain numbers with two significant figures: 99299299.22

The **fk** field creates a foreign key constraint to the table named in the fk field. For example, on row 3 of the arrest var_info metadata file the casenum column is set as a foreign key to the subject table. This indicates that all casenum values have a corresponding primary key value in the subject table.

The **label** field is the variable label that will be added to the SPSS segment database extract file. Rows that contain a value in the original field do not require a label as the original variables are not included in the database extract.

The **value labels** field is the variable value labels that will be added to the SPSS segment database extract file. The value labels should either have a value of NONE, indicating no value labels, a value of USE FK, which will load the value labels from the associated lookup table, or should be in a json list of list format. The json format should follow this convention `[[integer value 1, “description 1”], [integer value 2, “description 2”] ... [integer value n, “description n”]]` where n equals the total number of value labels.

The Excel files controlling the recidivism database table specification (recid_var_info.xls) work in the same manner as the var_info files with one exception, the **original** column was replaced with the **new** column. In the new column should be blank for all data that is loaded from the standardized database and set to TRUE for new column variables that are added going to be created on the recidivism database segment tables.

The business logic segment metadata and NLETS intermediate files the must be reviewed before initiating the stage phase of the conversion life cycle. The segment files need to match the specifications in the segment metadata files. The segment files are required to have the information currently listed in the segment metadata. Additional information can be provided in the NLETS files but this information must be reflected in the segment metadata files.

4.1.4 Flat File Metadata

Metadata files are also provided to control the specifications for the recidivism flat files. The following Excel metadata files are provided:

- 1) Criminal History Summary file
projectdir/metadata/summary/var_info.xls
- 2) Criminal History Flat file
projectdir/metadata/flat/var_info.xls
- 3) Demographic Flat file
projectdir/metadata/demographic/flat.xls

The Excel metadata file for the Criminal History Summary file works in the exact same manner as the segment metadata files. There is one row for each variable in the file and each row contains information regarding the column attributes (length, format, labels, value labels, etc.)

The Criminal History Flat and Demographic Flat files use the same layout as the segment and summary metadata files with one exception. In the Criminal History and Demographic Flat files the number of columns will be multiplied by the highest number of arrest cycles amongst all the subjects in the cohort. For example, if a cohort has a subject with 150 arrest cycles, then the columns present in the specification file will be multiplied by 150 times, so for the Criminal History Flat file this would mean that the file had 150*76 or 11,400 columns.

4.1.5 Metadata Updates

While an Excel metadata file is available to the user for manipulation, the conversion system loads the data from its json format. Therefore, in order for any changes made to the Excel metadata files to take effect the user must issue the Update Metadata command. When the user invokes this command all of the data contained in the Excel metadata files will be transformed and then saved into json format.

4.2 Business Logic – Coding Rules

4.2.1 Overview

CCHRRD Conversion System business logic coding rules provide the system with the ability to standardize arrest, court-sentencing and demographic records obtained from the NLETS parsing system. The business logic coding rule system is a hierarchical rule system following a structured directory format to accommodate the addition or removal of rules from the system. The coding rules support the following components listed from highest to smallest within the coding rule hierarchy: rule phases, rule managers, rule controllers, rulesets (generated by the Microsoft Excel crosswalk files), and rules. Before going into details of each of these components it is helpful to understand the overall structure of the business logic coding rules. The following diagram presents the coding rule structure, down to the rule phase level, for the three BJS cohorts currently processed through the system.

PROJECTROOT

```

|--- build (system default business logic stored here)
|--- metadata (default business logic metadata)
|--- rules (default business logic coding rules)
|--- project_data
|--- demo (PROJECTDIR)
|--- rules
|--- crosswalk_raw
|--- offense_crosswalks
|--- ori_crosswalks
|--- pre_process_rules_1
|--- pre_process_rules_2
|--- pre_process_rules_3
|--- process_rules_0
|--- process_rules_1
|--- process_rules_2
|--- process_rules_3
|--- process_rules_4
|--- process_rules_5
|--- sentence_crosswalks

```

When a project is created the coding rules are copied from the system default business logic and saved to *projectdir/rules* directory. Within this directory there are several folders; the *crosswalk_raw*, *offense_crosswalks*, *ori_crosswalk* and *sentence_crosswalks* folders will all be discussed in section 4.2 Business Logic – Coding Rules. The remaining folders, those starting with *pre_process* and *process*, represent the rule phases. Within the rule phase folders are a number of folders representing rule managers. Inside each rule manager is a rule controller file, named *func.py*, and a number of rulesets which contain one or more coding rules.

4.2.2 Rule Phases

Two rule phases are supported by the system; the pre-process phase and the process phase. Any directory starting with *pre_process* belongs to the pre-process rule phase and those starting with *process* belong to the process phase. The number at the end of the rule phase folder indicates the priority of the rule phase. The priority lets the system know what order the coding rules within the rule phase folders should be applied to the data when the pre-process or process commands are run. Using the above example when the pre-process command is invoked three phases of coding rules are applied; first the business logic coding rules in pre-process rule phase 1 get applied, then those in pre-process phase 2 and finally pre-process rule phase 3. The same pattern is followed in the process rule phases starting with process rule phase 0 and going to process rule phase 5.

Within a rule phase groups of rule managers are categorized by the segment they should be applied to. When the system runs the pre-process or process phase for a particular segment it will only load rules for

the given segment. For example in `pre_process_rules_1` there are four segment folders: `arrest`, `demographic`, `sentence`, and `supervision`.

```
|--- project_data
|--- demo (PROJECTDIR)
|--- rules
|--- pre_process_rules_1
|--- arrest
|--- demographic
|--- sentence
|--- supervision
```

If the process command is run for only the `arrest` segment:



Then only the rules managers in the `arrest` folder (bolded above) will be loaded for pre-process phase 1.

4.2.3 Rule Managers

Within rule segment rule phase there are one or more rule managers. A rule manager must be named after a variable in the segment it is processing and contain a rule controller and zero or more rulesets. In the below example the pre-process rule phase 2 contains four rule managers; `aori`, `arname`, `asource`, `astate`. The `astate` rule manager contains two rulesets (`crosswalk_final.txt` and `crosswalk_us.txt`) and a rule controller (`func.py`). When the system runs it will apply the rule manager to each row of segment data.

```
|--- project_data
|--- demo (PROJECTDIR)
|--- rules
|--- pre_process_rules_2
|--- arrest
|--- aori
|--- arname
|--- asource
|--- astate
|--- crosswalk_final.txt
|--- crosswalk_us.txt
|--- func.py
```

4.2.4 Rule Controllers

A rule controller is a python module file that must contain one of the following functions: `convert`, `assign_vals`, `translate`, `assign_id`, `validate`, or `cache`. While all these functions are supported the most commonly used function is `convert`. The `convert` function takes a value and row for parameters and returns a converted value. During application of business rules the rule controller will apply its function to each row of data. The function is passed the variable value corresponding to the rule manager name and the full row data. For the `convert` function the rule controller will return a converted value to the variable matching the rule manager name.

For example, the `astate` rule manager will be applied to each row in the arrest dataset. The rule controller function for `astate` is as follows:

```
|--- project_data
|--- demo (PROJECTDIR)
|--- rules
|--- pre_process_rules_2
|--- arrest
|--- astate
|--- crosswalk_final.txt
|--- crosswalk_us.txt
|--- func.py

def convert(value, row):
    convert_key = crosswalk['final'].metadata.get_key(row, as_str=True)
    missing = crosswalk['final'].missing or 99
    unknown = crosswalk['final'].unknown or 99

    if not convert_key:
        return missing
    try:
        if convert_key == 'us':
            return crosswalk['us'].convert(crosswalk['us'].metadata.get_key(row, as_str=True))
        else:
            return crosswalk['final'].convert(convert_key)

    except ConvertError as e:
        log.write('astate', [row['id'], row['rapstatex'], convert_key, e])
        return unknown
```

For each row the above rule controller will be passed the current `astate` value and the full row data. It will then apply the business coding logic and return the converted value to the `astate` variable.

4.2.5 Rulesets

A ruleset is python class named Crosswalk that will return zero or more output values based on an input key and the ruleset metadata. When the business logic coding rules are loaded during a conversion phase the rule manager creates a dictionary of Crosswalk objects, with dictionary keys set to the ruleset name and the value being the Crosswalk object, and makes this object available to the rule controller. A rule controller can reference the ruleset directly using the crosswalk object, as seen in the example above `crosswalk['final'].convert(convert_key)`, or it can be referenced by a helper function. The a/cncic rule controllers (offense coding) use the `convert_offenses` function and the snttextx rule controllers (sentencing coding) use the `convert_sentences` function.

While a ruleset uses metadata rules to provide specifications for how the ruleset should conduct rule coding (see 4.5.1 for information on metadata rules), all results will adhere to the following behavior. First, a ruleset will use one or more variables from the record it is being applied to as the input key. The second step is to determine if the rules within the ruleset are able to handle the input key. For exact match rules the input key is checked against an indexed list of all rules with the same rule priority (priority is the order at which a rule is evaluated in the ruleset) to see if the input key matches a rule key. If a match is found rule coding halts and the output values for that rule are assigned to the output variables. If regular expression rules are present and a match was not found in the exact match rules then regular expression key pattern is checked against the input key to see if the rule pattern can handle the input key. Each key pattern is evaluated until a match is found at which point the output values are assigned to the output variables.

4.2.6 Coding Rules

Within a ruleset are one or more coding rules. The coding rules will either provide the ruleset with instructions on how to operate, known as metadata rules, or provide a key/value type matching rule, known as a conversion rule. The ruleset (Crosswalk object class) currently supports 15 types of metadata rules and 36 types of conversion rules. As of writing this manual there are over 1 million coding rules in the conversion system.

4.3 Pre-Process Rule Phase

4.3.1 Making Updates

While the pre-process rule phase consists of 28 rule managers spread across three rule phases only the rulesets within the afed and cfed rules should require updates when processing a new cohort. The afed and cfed rule managers determine if an arrest or sentence record is a federal record. Additionally, the cfed rule will fill in the cstate variable if it is missing based on the corresponding arrest record's astate

value. Since the fed rules are the only pre-process rules that require updating the conversion system provides an update process complete with a coding interface to provide a more user friendly method of making updates.

When working on a new cohort fed updates should be made after the data staging and pre-process phases have been run. After the initial pre-process phase is complete the fed coding reports should be reviewed two reports are created for each segment. The files are outputted to:

Arrest

- ▶ *projectdir/output/report/pre_process/afed.xls*
- ▶ *projectdir/output/report/pre_process/afed_unknown.xls*

Sentence

- ▶ *projectdir/output/report/pre_process/cfed.xls*
- ▶ *projectdir/output/report/pre_process/cfed_unknown.xls*

After reviewing these files for accuracy on coding a/cfed based on the ori, updates should be made to the *projectdir/rules/ori_crosswals/ORI_Rules.xlsx* file for any inaccurate assignment of the a/cfed value.

The ORI_Rules file provides a user friendly interface to making updates to the ORI rulesets. After making updates, the updates must be applied by running the *update_rules* command and selecting 1 for “Load Pre-Process Rules” at the first prompt.

The ORI_Rules Excel crosswalk workbook contains one worksheet for each ruleset; ori, name, pos1, pos2, embedd. The worksheets are split up into several columns while the columns vary by worksheet, each will have a *rule_type* column, a *key* column, a *value* column, and a *priority*. The *rule_type* field supports *exact* rule types (look for an exact match on the input key) and any regular expression rule type (see XX for more information on regular expression rules). The *key* column, identified by “key:” in the column header, specified the key value that should be matched on. The *value* column, identified by “value:” in the column header, specifies the output value that should be given to the a/cfed variable. The *priority* is the order the rules should be checked, *exact* rules should have a priority of 1, as this allows for indexed searching on all *exact* rules which increases the performance of the system. An additional column, *AgencyName*, is present in most of the ORI crosswalks. This column is a convenience column and has no impact on the resulting ruleset; thus it can be left blank or filled out when making ORI rule updates.

The first worksheet, ori, is the most commonly updated worksheet. This worksheet attempts to code the a/cfed variable using the a/cori field as the input key. When a new ORI value is detected in the reports it

should be added to the key:ori column with its corresponding fed output value in the value:fed column. The rule_type should be *exact*, the priority should equal 1.

The second worksheet, *name*, will attempt to code the a/cfed variable using the agency name (arrname/crtname) as the input key. The third worksheet, pos1, will attempt to code the a/cfed value using the 3rd-5th characters in the a/cori variable as the input key. The fourth worksheet, pos2, will attempt to code the a/cfed value using the 8th and 9th characters of the a/cori variable as the input key. The fifth worksheet, embed, will attempt to code the a/cfed variable by using a regular expression search on the agency name. For this worksheet a regular expression rule type should be used and the priority should increment.

When making updates to the crosswalk it is important to keep in mind that the rule controller will determine which crosswalk to use based on the state of a record. Below is the logic that the rule controller follows when deciding which crosswalk to use.

A/CFED Rules

- 1) IF ori_invalid flag is False
 - a. Try to code a/cfed using a/cori on ORI ruleset
 - b. Try to code a/cfed using a/cori characters 3-5 on the ORI_POS1 ruleset
 - c. Try to code a/cfed using a/cori characters 8-9 on the ORI_POS2 ruleset
- 2) IF ori_invalid flag is True
 - a. Try to code a/cfed using arr/crtname on NAME ruleset
 - b. Try to code a/cfed using arr/crtname on EMBED ruleset
- 3) IF a/cfed is not coded after step 2
 - a. Recover data from arrest file. Match on first entry of CASESNUM and CYCNUM and fill in cfed based on afed

If CSOURCE=1 and CSTATE=99 also bring back assign ASTATE value to CSTATE
 - b. If recovery not possible
 - i. IF CSOURCE = 2 code CFED=2
 - ii. IF CSOURCE not equal 2 OR RAPSTATE not a valid state code CFED=9
- 4) All cases coded via 1b, 1c, 2b, 3a or 3b should be added to report and reviewed and oris added to crosswalks as appropriate.

4.4 Process Rule Phase

4.4.1 Making Updates

When working on a new cohort offense updates should be made after the data staging, pre-process, and process phases have been run. The Process phase has two primary types of rule coding; the offense rules which code a series of variables that provide standardized information related to the type and severity of offense for an arrest or court-sentencing record, and the sentence rules which code a series of variables that provide information on the type and length of the sentence associated with a court-sentence record.

4.4.1.1 Offense Rules

After the initial process phase is complete offense coding report files should be reviewed. The initial run will output report files for each segment. Since the arrest and sentence offense rules are the same and there are more arrest records than court-sentencing records the arrest reports should be used for initial review. The files are outputted to *projectdir/output/review/set/arrest/all/round1*. The ancic and ancic_wvfbi files are the most important files to review as they provide all records that were not able to be coded for the offense flag variables (ancic, achg, ainc, adom, aweap, acdv, and apg). The adsp report can also be reviewed but it may contain information that does not require actual updates to the adsp rules. A full review of the arrest.sav and sentence.sav are required to detect all records that require offense coding. After reviewing the report and database extract SPSS files all states that require updates should be run through the process phase using the region parameter. After running the individual regions, report files will be available for each region (*projectdir/output/review/set/arrest/region/round*). All missing input keys from the ancic report should be added to the state crosswalks and all missing input keys from the ancic_wvfbi report should be added to the wvfbi crosswalks. Additionally, the SPSS database extract files for these regions should be reviewed and updates made to the a/cchgsvr, a/cinc, a/cdsp, and nonarr/noncrt worksheet files as needed.

Due to the complexity with updating all these rules an offense crosswalk Excel workbook was created for each region for both the state records and WVFBI records. The regular state offense crosswalks are located at *projectdir/rules/offense_crosswalks/regular_state* and the WVFBI crosswalks at *projectdir/rules/offense_crosswalks/wvfbi*. Each region has its own crosswalk, including a WVFBI crosswalk for federal records (*projectdir/rules/offense_crosswalks/wvfbi/fed.xls*).

State Crosswalks

Within the *projectdir/rules/offense_crosswalks/regular_state* folder are a series of Excel workbooks, two for each region. The files are named by the region abbreviation (ak for Alaska offense crosswalk) and region abbreviation followed by pred (ak_pred for Alaska prediction crosswalk). The prediction

crosswalk should be ignored as they are a deprecated prediction approach that is disabled in the current version. Each offense crosswalk workbook contains a series of worksheets. The number worksheets differ for each state but may include; *offense* for coding the offense flag variables, *achgsvr/cchgsvr* for charge severity coding, one or more *adsp/cdsp* for disposition coding, *ainc/cinc* for additional inchoate coding, *adom/cdom* for additional domestic violence coding, and *nonarr/noncrt* for coding the nonarr/noncrt variable.

Each crosswalk worksheet must contain at least one key column and one value column. The key column(s) can be identified by a column header string of “key:” followed by the name of the variable that will be used to construct the input key. For example a key column of “key:cstatnumx” will use the record’s cstatnumx variable as the input key. The value column(s) have a column header of “value:” followed by the variable that rule will output to. For example, a value column of “value:adom” will result in the rule outputting the rule value to the arrest domestic violence variable. All of the worksheets allow for more than one key and value columns. If more than one key column is provided than the input key will be the concatenation of the input values. If more than one value column is provided than a match on the rule will return all the value columns as a list of values. In addition to the key and value columns a rule_type column and priority column are available but are not required. The rule_type field supports any regular expression rule type (see XX for more information on regular expression rules). The priority is the order the rules should be evaluated, if no priority is provided the rule will be assigned a priority of 1 in the ruleset.

Example 1: In the below example we have the partial California cdsp1 crosswalk. In the below crosswalk there is one key column and one value column. The key column, “key:cdisptextx” instructs the ruleset to use the cdisptextx variable as the input key. The value column, “value:cdsp” instructs the ruleset to output matched rules to the cdsp variable. When cdsp rule manager is invoked it will look at each records cdisptextx variable and try to find a match on the three rules in ruleset.

key:cdisptextx	value:cdsp
ARRESTEE EXONERATED	11
COUNSELED AND RELEASED	11
EXTRADITED	14

Example 1:

If a record contains a *cdisptextx* value of “counseled and released” it’s cdsp value will be assigned a value of 11.

Example 2:

If a record contains a *cdisptextx* value of “extradited” it’s cdsp value will be assigned a value of 14.

Example 3:

If a record contains a *cdisptextx* value of “not extradited” it’s *cdsp* value will not be assigned a value.

Special cases

- 1) The offense worksheet value columns are not the full variable names for the offense flag variables. Since the offense worksheet creates the arrest and court-sentencing rulesets the “a” and “c” are removed from the value column header.
- 2) The offense and disposition crosswalks can be named offense1, offense2, ... offenseN or cdsp1, cdsp2, ..., cdspN, etc where N is the total number of crosswalks. When the crosswalks are numbered they will be applied in order from 1 to N until a match is found on one of the crosswalks.

For example, if there are two *cdps* crosswalks, *cdsp1* and *cdsp2*, then all the rules from *cdsp1* will be checked first and if the rules are not able to handle the input key then the rules in *cdsp2* will be applied.

- 3) The offense crosswalks support a special remove crosswalk. The remove crosswalk is named as offenseN_remove where N is the priority for removal. The remove crosswalk will remove the matched key from the input string and replace it with the value in the value column.

For example the ca offense workbook has three offense crosswalks; offense1, offense2_remove, and offense3. When these are applied the rules within offense1 will try to handle the input key if none of the rules can handle the key it will be sent to offense2_remove. In the remove crosswalk each rule will attempt to handle the input key, if a match is found the key value will be removed from the input key. Then the transformed input key will be sent to offense3 for coding. See ca.xls for example.

- 4) The offense worksheet can be replaced with ancic and cncic worksheets if different arrest and court-sentencing rules are needed for coding the offense flag variables. If these sheets are added the ancic worksheet will code the arrest offense flag variables and the cncic will code the court-sentencing offense flag variables. The ancic and cncic worksheets behave in the same manner as the offense worksheets.

- 5) If a rule type column is not included in the crosswalk it is still possible to create a regular expression rule. Simply place the regular expression rule identifier in the first key column. For example, consider a adsp crosswalk that contains one rule that assigns 35 to adsp if the adisptextx variable is an exact match to “convicted”.

key:adisptextx	value:adsp
convicted	35

If the user wants to modify this rule to output the 35 to adsp if the string convicted is located anywhere in the string they can simply add :re:search: to the key column.

key:adisptextx	value:adsp
:re:search:convicted	35

4.4.1.2 Sentencing Rules

Within the court-sentencing segment there are a series of sentencing variables (cnf, incmax, incmin, suspmax, suspmin, prb) that require a very complex series of rulesets to achieve complete and accurate coding. Each state has its own set of rulesets to accommodate this complex coding. In an effort to increase the efficiency of sentencing ruleset creation a series of Excel crosswalks were made for each state and WVFBI (located in the *projectdir/rules/sentence_crosswalks* directory). In the *sentence_crosswalks* directory is a folder for each state identified by the state abbreviation, the crosswalk used for a particular record will use the state rulesets matching its rapstatex variable. Each state folder has one Excel crosswalk file with the exception of the wvfbi folder, which has two Excel files. However, the WVFBI crosswalks are the least complex of all of these rules, but most time consuming to update.

The WVFBI crosswalks contain a 2_or_more crosswalk and a single crosswalk. The 2_or_more crosswalk is intended to match cdisptextx values (the key column) that are present in multiple court-sentencing records, while the single crosswalk contains all cisptextx values that appear in only one record. After running the process rules the user should review the full sentence database extract and extract all records where incmax = None. Each of these records will need to be added to the 2_or_more or single crosswalk. The two or more crosswalk uses a regular expression search (:re:search:) on the key column value and return the value columns to the output variables. Any # sign present in the key column will match a repeating digit group. The single crosswalk uses an exact match search and is much more efficient in processing time.

The state Excel crosswalk workbook contains a number of worksheets. Each worksheet starts with a number and is followed by `_rules` or `_metadata`. The `#_rules` and `#_metadata` crosswalks will be combined to produce one ruleset for each `#` present in the workbook, so if there is `1_rules`, `1_metadata`, `2_rules`, and `2_metadata` the result will be the creation of two rulesets (`ruleset_1` and `ruleset_2`). The `_rules` worksheet provides all the regular expression rules used in the ruleset and the `_metadata` lists all the metadata rules for a ruleset. The `_rules` worksheet contains a `ruletype` column, a `key` column, one or more sentencing variable columns, and a `priority` column., the `key` is either a regular expression pattern or a plaintext pattern that the rules will attempt to match in the SNTTEXT data, the sentence variable columns contain the output values or formulas to assign to each of the sentence variables, and finally the rule `priority` column contains the order that the coding rules should be applied (see regular expression rules below for more details).

Rule Type

The rule type provides the ruleset with information on how the rule should be evaluated during processing. Each rule type has a unique rule coding behavior, constraints, and functionality. See section 4.5.2 for a complete listing of all available rule types.

Rule Key

The rule key is a pattern that is used to check if rule is able to “handle” the input key data (generally SNTTEXT variable for sentence crosswalks). The term handle means that the rule key pattern finds a match to the input key data. If a match is found the rule is able to handle the input key data, and this results in the output values being assigned to the output variables. The sentence regular expression rules support two types of rule key patterns, a plaintext key and a regular expression key. Details on the plaintext key pattern can be found below. The regular expression pattern is only available when using the Regex rule types (see section 4.5.2.16).

Plaintext Key Pattern

Plaintext key pattern allows a user to input the text as they see it in the arrest or court-sentencing data. The plaintext key will be escaped on all python regular expression characters, allowing for the user to not have to understand Python regular expressions in order to create regular expression rules. The plaintext pattern also supports the capturing of repeating digit groups by transforming the “#” character into a regular expression digit capture pattern. The easiest way to illustrate a plaintext key is through an example.

Let's assume that a user wants to create a regular expression pattern that will extract the number of years incarceration time and the number of years probation time from each record that has the following string in the `snttext` variable:

`snttext` data: "sentence to 10 years in prison, \$100 dollars in fines, 5 years probation"

A regular expression pattern to match the above and capture the digit groups for output formulas would look like this:

"sentence to `(\d+)` years in prison, `\$(\d+)` dollars in fines, `(\d+)` years probation"

Within the above string the digit groups were removed and replaced with `(\d+)` and dollar sign was changed to `\$`. Within the Python regular expression language a `"\d"` indicates a digit character, a plus sign `"+"` matches one or more instances of the preceding strings, a `$` indicates the end of the string so it needed to be escaped with a backslash `"\"`.

While the regular expression patterns are very powerful, like any other language they take time and practice to be proficient in them. To accommodate the user, the `plaintext` key abstracts the regular expression language while still allowing a user to capture repeating digit groups for use in output formulas.

The above regular expression in plaintext format:

"sentence to `#` years in prison, `$#` dollars in fines, `#` years probation"

Since the `plaintext` key is escaped by the system the regular expression keywords (i.e. the dollar sign `"$"`) no longer need to be escaped by the user. The repeating digit groups are captured using a pound sign `"#"`. When the system sees the `#` it translates this into `(#|#\d+|\d+)`, which means match the `#` sign, or match the `#` sign followed by one or more digits, or match one or more repeating digits.

After the rule key is created, using either a regular expression key or a `plaintext` key, the user can use the captured repeating digit groups in an output formula. Continuing with the above example, if the user wanted to output the first group to the `incmax` field and the third group to the `prb` field the full crosswalk rule would look like this:

Ruletype	Key	cnf	incmax	incmin
:re:search:	sentence to # years in prison, \$# dollars in fines,	1	99299299.22	99899899.88

years probation

And the translated ruleset format rule would look like this:

```
re:search:sentence to (##\d+|\d+) years in prison, \$(##\d+|\d+) dollars in fines, (##\d+|\d+) years
probation\t1|99299299.22|99899899.88\t1
```

Sentencing Output Variables

The sentencing output variables consist of cnf, incmax, incmin, suspmax, suspmin, and prb. These values capture the type of sentence, the length of the sentence, the length of a suspended sentence, and the length of probation time. The information is extracted from the SNTTEXT field and these columns provide the ability to code this data with either direct values (99299299.22 for life) or using a formula to take the extracted data and code an amount of time (365 for “sentenced to 1 year in prison”).

Rule Priority

Within the sentence crosswalk rule worksheets there is a column called “priority”. The priority column indicates the order of the rules. A regular expression rule is evaluated to see if the rule key pattern can handle the input key (SNTTEXT data for all sentence crosswalks). A ruleset will evaluate all regular expression rules until a rule key pattern can handle the input key starting with the highest priority (smallest integer value) and going to the lowest priority (highest integer value).

4.5 Rule Reference

4.5.1 Metadata Rules

The rulesets support a number of metadata rules that provide the ruleset with instructions on how to preform rule coding. These rules are added to the _metadata tab.

4.5.1.1 Key Metadata Rules

The key metadata rules provide instructions to the ruleset on how to create and process the input key.

4.5.1.1.1 Keys

The keys metadata rule allows the user to set the variable(s) that will be used to create the input key. To declare a keys rule place :keys: in the field column and a bar delimited list of variable names in the value1 column.

Rule Type

:keys:

Usage

SNTTEXT Data = “Sentenced to 10 years in prison”

CDISPTEXT Data = “2 years probation”

Metadata Crosswalk Format

field	value1	value2	value3	value4	value5
:keys:	snttextx cdispstextx				

Ruleset/Machine Format

:keys: snttextx|cdispstextx

Result

The above rule will instruct the ruleset to use the snttextx and cispstextx variables to create the rule input key. The resulting input key will be “sentenced to 10 years in prison|2years probation”

4.5.1.1.2 Key Transformation

The key metadata allows the user to create a key transformation rule. There are three types of key transformation rules; a replace transformation, a substitution transformation, and a predicate replace. For each of the three key transformation rules types the *:key:* should be placed in the field column, the variable to do the replacement on should be in the value1 column, the key transformation rule type should be in the value2 column. The *:replace:* rule type should have the string that is to be replaced in value3 and value4 should have the replacement string. The *:sub:* rule type behaves the same as the *:replace:* but allows for a regular expression pattern in the value3 field. The predicate replace has the a regular expression pattern that searches for a particular string in the value3 field, a second regular expression pattern that will match the text to be replaced in the value4 field, and the value5 field is the replacement text.

Rule Type

:key: :replace:

:key: :sub:

:key: :predicate_re_replace:

Usage***Replace Key Transformation***

SNTTEXT Data = “Sentenced to 10 years in prison 5 years prab”

Metadata Crosswalk Format

field	value1	value2	value3	value4	value5
-------	--------	--------	--------	--------	--------

:key:	snttextx	:replace:	prab	probation
-------	----------	-----------	------	-----------

Ruleset/Machine Format

:key: snttextx :replace: prab probation

Result

The above rule will instruct the ruleset to look for the prab string in the use the snttextx variable and replace all instances of “prab” with “probation” within the input key. The resulting input key will be “sentenced to 10 years in prison 5 years probation”.

Substitution Key Transformation

SNTTEXT Data1= “Sentenced to 10 years in prison 5 years prob”

SNTTEXT Data2 = “Sentenced to 10 years in prison 5 years prab”

Crosswalk Format

field	value1	value2	value3	value4	value5
:key:	snttextx	:sub:	prob prab	probation	

Ruleset/Machine Format

:key: snttextx :sub: prob|prab probation

Result

The above rule will instruct the ruleset to look for the regular expression that matches prob|prab (match prob or prab) in the snttextx variable and replace all instances of the regular expression pattern prab” with “probation”. The resulting input key for both SNTTEXT Data1 and 2 will be “sentenced to 10 years in prison 5 years probation”.

Predicate Replace Transformation

SNTTEXT Data1= “Sentenced to 10 years in prison 5 years prab”

SNTTEXT Data2= “10 years in prison 5 years prab”

Crosswalk Format

field	value1	value2	value3	value4	value5
:key:	snttextx	:sub:	sentenced	prab	probation

Ruleset/Machine Format

```
:key:  snttextx      :predicate_re_replace:  sentenced      prab      prob
```

Result

The above rule will instruct the ruleset to first look for the regular expression pattern in the value3 field. If it finds a match on that field than replace the regular expression in the value4 field with the value5 field. In the above example SNTTEXT Data 1 will find a match on the “sentenced” pattern and then prab will be replaced with probation. The resulting input key for SNTTEXT Data1 will be “sentenced to 10 years in prison 5 years probation”. For SNTTEXT Data 2 the sentenced string is not present in the input key so no transformation will be performed. The resulting input key for SNTTEXT Data 2 will be “sentenced to 10 years in prison 5 years prab”.

4.5.1.1.3 Key Predicate

The key predicate metadata will check to see if the input key matches a regular expression pattern. If the input key matches then rule coding continues, if there is no match then rule coding is terminated for that record. To declare a key predicate rule place `:key_predicate:` in the field column and a regular expression pattern in the value1 column.

Rule Type

```
:key_predicate:
```

Usage

SNTTEXT Data 1= “Sentenced to 10 years in prison”

SNTTEXT Data 2= “2 years probation”

Metadata Crosswalk Format

Field	value1	value2	value3	value4	value5
:key_predicate:	sentenced				

Ruleset/Machine Format

```
:key_predicate:      sentenced
```

Result

The above rule will instruct the ruleset to check for the regular expression pattern “sentenced” in the input key. If a match is found to “sentenced” then rule coding will continue as normal otherwise coding is terminated for that record. The SNTTEXT Data 1 field finds a match to “sentenced” so it will continue through the rule coding process. The SNTTEXT Data 2 does not contain a match to “sentenced” so rule processing will be terminated.

4.5.1.1.4 Remove Key

The remove key metadata will instruct the ruleset to behave differently than all other rulesets. Instead of attempting to code output values the remove key tell the ruleset that it should iterate over all rules and remove matched rule key patterns from the input key. To declare a remove key rule place `:remove_key:` in the field column and True in the value1 column.

Rule Type

`:remove_key:`

Usage

SNTTEXT Data 1= “Sentenced to 10 years in prison 25 years in ps 10 year in conf”

Metadata Crosswalk Format

Field	value1	value2	value3	value4	value5
<code>:remove_key:</code>	True				

Rule Crosswalk Format

Ruletype	Key	Value
<code>:re:search:</code>	<code>\d+ years in ps</code>	<code>:none:</code>
<code>:re:search:</code>	<code>\d+ years in conf</code>	<code>:none:</code>

Ruleset/Machine Format

`:remove_key: True`

Result

The above rule will instruct the ruleset to remove all handled rules from the input key. In the above example the SNTTEXT Data 1 field finds a match on “`\d+ years in ps`” so it will replace that occurrence from the input string with the output value, which in this case is `:none:` indicating that it should be removed. The second rule looks for “`\d+ years in conf`” and removes matched results. After this ruleset has finished processing the input key will be “sentenced to 10 years in prison” rather than the former “Sentenced to 10 years in prison 25 years in ps 10 year in conf”.

4.5.1.1.3 Null Key Predicate

The null key predicate metadata rule will check to see if the input key **DOES NOT** match a regular expression pattern. If the input key does not match then rule coding continues, if there is a match then rule coding is terminated for that record. To declare a null key predicate rule place `:null_predicate:` in the field column and a regular expression pattern in the value1 column.

Rule Type

:key_predicate:

Usage

SNTTEXT Data 1= “Sentenced to 10 years in prison”

SNTTEXT Data 2= “2 years probation”

Metadata Crosswalk Format

Field	value1	value2	value3	value4	value5
:null_predicate:	sentenced				

Ruleset/Machine Format

:key_predicate: sentenced

Result

The above rule will instruct the ruleset to check for the regular expression pattern “sentenced” in the input key. If a match is found to “sentenced” then rule coding will **not** continue otherwise coding will be continued as normal for that record. The SNTTEXT Data 1 field finds a match to “sentenced” so rule coding will be terminated. The SNTTEXT Data 2 does not contain a match to “sentenced” so rule processing will continue as normal.

4.5.1.2 Variable Metadata Rules

The key metadata rules provide instructions to the ruleset on how to create and process the output variables.

4.5.1.2.1 Vars

The vars metadata rule allows the user to set the output variable(s) that will be coded when a rule is able to handle the input key. To declare a vars rule place :vars: in the field column and a bar delimited list of variable names in the value1 column.

Rule Type

:vars:

Usage**Metadata Crosswalk Format**

field	value1	value2	value3	value4	value5
:vars:	incmax incmin				

Ruleset/Machine Format

```
:vars:  incmax|incmin
```

Result

The above rule will instruct the ruleset to have a rule code the incmax and incmin variables when a rule is able to handle the input key.

4.5.1.2.2 Lookup

The lookup metadata rule allows the user to do a referential integrity check on each rule output value. The lookup rule will check that each row output value is contained within a lookup table. This check ensures that no referential integrity errors will occur when saving to the segment database tables. When the lookup validation fails for a rule it will be added to the referential integrity rule report (see section 5.5). Since the lookup rule uses the vars metadata must contain the same number of variables and preserve the same variable order as the vars metadata rule. To declare a lookup rule place `:lookup:` in the field column and a bar delimited list of table names in the value1 column.

Rule Type

```
:lookup:
```

Usage**Metadata Crosswalk Format**

field	value1	value2	value3	value4	value5
:vars:	cdsp cdom				
:lookup:	dsp dom				

Ruleset/Machine Format

```
:vars:      cdsp|cdom
:lookup:    dsp|dom
```

Result

The above rule will instruct the ruleset to check that all the cdsp and cdom output variables do not violate referential integrity on the dsp and dom tables (respectively). If a coding rule does has a cdsp or cdom value that does not appear in the dsp or dom table then this will be logged to a report.

4.5.1.2.3 Missing

The missing metadata rule allows the user to set the default missing output value for a ruleset. Missing values are assigned when the input key is blank. Since the missing rule uses the vars metadata it must contain the same number of variables and preserve the same variable order as the vars metadata rule. To

declare a missing rule place *:missing:* in the field column and a bar delimited list of variable output values in the value1 column.

Rule Type

:missing:

Usage

Metadata Crosswalk Format

field	value1	value2	value3	value4	value5
<i>:vars:</i>	incmax incmin				
<i>:missing:</i>	99999999.99 99999999.99				

Ruleset/Machine Format

```
:vars:          incmax|incmin
:missing:      99999999.99|99999999.99
```

Result

The above rule will instruct the ruleset to set the default missing value for incmax and incmin to 99999999.99 and 99999999.99.

4.5.1.2.4 Unknown

The unknown metadata rule allows the user to set the default unknown output value for a ruleset.

Unknown output values are assigned when all rules within a ruleset are unable to handle the input key.

Since the unknown rule uses the vars metadata it must contain the same number of variables and preserve the same variable order as the vars metadata rule. To declare an unknown rule place *:unknown:* in the field column and a bar delimited list of variable output values in the value1 column.

Rule Type

:unknown:

Usage

Metadata Crosswalk Format

field	value1	value2	value3	value4	value5
<i>:vars:</i>	incmax incmin				
<i>:unknown:</i>	99999999.99 99999999.99				

Ruleset/Machine Format

```
:vars:          incmax|incmin
:unknown:      99999999.99|99999999.99
```

Result

The above rule will instruct the ruleset to set the default unknown value for incmax and incmin to 99999999.99 and 99999999.99.

4.5.1.2.5 Skip Missing

The skip missing metadata rule tells the crosswalk that it should not assign missing values to output variables when the input key is blank. By default the system will assign the default missing values when no input key is provided. To declare a skip missing rule place *:skip_missing:* in the field column and a bar delimited list of variable names in the value1 column.

Rule Type

:skip_missing:

Usage**Metadata Crosswalk Format**

field	value1	value2	value3	value4	value5
:vars:	incmax incmin				
:skip_missing:	incmax				

Ruleset/Machine Format

```
:vars:          incmax|incmin
:skip_missing:  incmax
```

Result

The above rule will instruct the ruleset to set the default missing value for the incmin variable but to **not** output a missing value for the incmax variable.

4.5.1.2.6 Skip Unknown

The skip unknown metadata rule tells the crosswalk that it should not assign unknown values to output variables. By default the system will assign the default unknown values when no rules within the ruleset are able to handle the input key. To declare a skip unknown rule place *:skip_unknown:* in the field column and a bar delimited list of variable names in the value1 column.

Rule Type

:skip_unknown:

Usage**Metadata Crosswalk Format**

field	value1	value2	value3	value4	value5
-------	--------	--------	--------	--------	--------

:vars:	incmax incmin
:skip_unknown:	incmax

Ruleset/Machine Format

```
:vars:          incmax|incmin
:skip_unknown:  incmax
```

Result

The above rule will instruct the ruleset to set the default unknown value for the incmin variable but to **not** output an unknown value for the incmax variable.

4.5.1.2.7 Overwrite

The overwrite metadata rule tells the crosswalk that it should overwrite existing output variables if they already have a value. By default the system will not overwrite existing data. To declare an overwrite rule place `:overwrite:` in the field column and a bar delimited list of variable names in the value1 column.

Rule Type

```
:overwrite:
```

Usage**Metadata Crosswalk Format**

field	value1	value2	value3	value4	value5
:vars:	incmax incmin				
:overwrite:	incmax				

Ruleset/Machine Format

```
:vars:          incmax|incmin
:overwrite:     incmax
```

Result

The above rule will instruct the ruleset to overwrite incmax output variables but to **not** overwrite the incmin variable.

4.5.2 Regular Expression Rules

A regular expression is a sequence of characters that forms a search pattern that is used to match some text based on the supplied pattern. <http://docs.python.org/2.7/library/re.html> The regular expression coding rules allow a user to do more complex pattern matching and extract values from matched patterns to perform arithmetic based functions.

Math Rules

The regular expression math rules enable mathematical formulas in the sentencing variable output values. Not only are formulas allowed, but the math rules provide a method for dynamically referencing data from a record's SNTTEXT field within the output formula. For example, a user can use the number of years in the SNTTEXT data to calculate the incmax output variable. This is accomplished by creating digit groups within the regular expression pattern and referencing these groups using a digit group placeholder in the sentence output value. A digit group is created in the key pattern using a regular expression group with a repeating digit expression (\d+), and these groups are referenced in the output value using “#n” where n is the index of the repeating digit expression. A more user friendly approach is available to create digit groups; this approach is the plaintext key pattern (see below for more details).

The following are all the regular expression rules that are currently supported within the Excel Sentence Crosswalks.

4.5.2.1 Search

The search rule transforms the user defined plaintext key into a regular expression pattern which will attempt to match the given plaintext key **anywhere** within the SNTTEXT data and if a match is found assign the provided sentence output values.

Rule Type

:re:search:

Usage

SNTTEXT Data = “Sentenced to life in prison”

Rule Crosswalk Format

Ruletype	Key	cnf	incmax	incmin
:re:search:	life	1	99299299.22	99899899.88

Ruleset/Machine Format

```
:re:search:life 1|99299299.22|99899899.88 1
```

Result

The above rule will match the string “life” in the SNTTEXT data and code the following output variables:

cnf = 1 incmax = 99299299.22 incmin = 99899899.88

4.5.2.2 Match

The match rule transforms the user defined plaintext key into a regular expression pattern which will attempt to match the given plaintext key **at the beginning** of the SNTTEXT data and if a match is found assign the provided sentence output values.

Rule Type

:re:match:

Constraints

- 1) Key pattern must match at start of the SNTTEXT data

Usage

SNTTEXT Data 1 = “Sentenced to life in prison”

SNTTEXT Data 2 = “life in prison”

Rule Crosswalk Format

Ruletype	Key	cnf	Incmax	incmin
:re:match:	Life	1	99299299.22	99899899.88

Machine Format

```
:re:match:life 1|99299299.22|99899899.88 1
```

Result

The above rule will not match SNTTEXT data 1 because the SNTTEXT data 1 string starts with “Sentenced” but it will match key data 2 because it starts with “life”. The following sentence output values will be assigned to the case corresponding to SNTTEXT data 2:

cnf = 1 incmax = 99299299.22 incmin = 99899899.88

4.5.2.3 Exact

The exact rule transforms the user defined plaintext key into a regular expression pattern which will attempt to match the given plaintext key **to the entire SNTTEXT data field** and if a match is found assign the provided sentence output values.

Rule Type

:re:exact:

Constraints

- 1) Key pattern must entire SNTTEXT field

Usage

SNTTEXT Data 1 = “Sentenced to life in prison”

SNTTEXT Data 2 = “life in prison”

Rule Crosswalk Format

Ruletype	Key	cnf	Incmax	incmin
:re:exact:	life in prison	1	99299299.22	99899899.88

Machine Format

```
:re:exact:life in prison  1|99299299.22|99899899.88      1
```

Result

The above rule will not match SNTTEXT data 1 because the plaintext key does not match the entire SNTTEXT data 1 string but it will match SNTTEXT data 2 because the plaintext key pattern “life in prison” matches the entire SNTTEXT data 2 field “life in prison”. The following sentence output values will be assigned to the case corresponding to SNTTEXT data 2:

```
cnf = 1          incmax = 99299299.22          incmin = 99899899.88
```

4.5.2.4 End

The end rule transforms the user defined plaintext key pattern into a regular expression pattern which will attempt to match the given plaintext key **from some point in the SNTTEXT data field to the end of the SNTTEXT data field** and if a match is found assign the provided output values.

Rule Type

```
:re:end:
```

Constraints

- 1) Key pattern must match to the end of the SNTTEXT data

Usage

SNTTEXT Data 1 = “Sentenced to life in prison and no probation”

SNTTEXT Data 2 = “Sentenced to life in prison”

Rule Crosswalk Format

Ruletype	Key	Cnf	Incmax	incmin
:re:exact:	life in prison	1	99299299.22	99899899.88

Machine Format

```
:re:exact:life in prison 1|99299299.22|99899899.88 1
```

Result

The above rule will not match SNTTEXT data 1 because the plaintext key does not match to the end of the SNTTEXT data 1 field but it will match SNTTEXT data 2 because the key “life in prison” matches to the end of the SNTTEXT data 2 field “Sentence to life in prison”. The following output values will be assigned to the case corresponding to SNTTEXT data 2:

```
cnf = 1          incmax = 99299299.22          incmin = 99899899.88
```

4.5.2.5 Tilde

The tilde rule transforms the user defined plaintext key into a regular expression pattern which will attempt to match the given plaintext key **from either the start of the SNTTEXT string or a tilde to the end of the string or a tilde** and if a match is found assign the provided output values.

Rule Type

```
:re:tilde:
```

Constraints

- 1) Key pattern must match at start of the SNTTEXT data or by preceded by a tilde
- 2) Key pattern must match to the end of the SNTTEXT data or by proceeded by a tilde

Usage

SNTTEXT Data 1 = “life in prison”

SNTTEXT Data 2 = “no court fees~life in prison”

SNTTEXT Data 3 = “life in prison~no court fees”

SNTTEXT Data 4 = “no probation~life in prison~no court fees”

SNTTEXT Data 5 = “sentenced to life”

SNTTEXT Data 6 = “no court fees~sentenced to life in prison”

SNTTEXT Data 7 = “sentenced to life in prison~no court fees”

SNTTEXT Data 8 = “no probation~sentenced to life in prison~no court fees”

Rule Crosswalk Format

Ruletype	Key	cnf	Incmax	incmin
:re:tilde:	life in prison	1	99299299.22	99899899.88

Machine Format

```
:re:tilde:life in prison 1|99299299.22|99899899.88 1
```

Result

The above rule will match on SNTTEXT data 1 – SNTTEXT data 4, because the plaintext key matches “life in prison”, “~life in prison”, “~life in prison~”, “life in prison~” from either the start of the SNTTEXT string or a tilde to the end of the string or a tilde. The above tilde rule does match “life in prison” on SNTTEXT data 5 – SNTTEXT data 8 because “life in prison” does not appear at the beginning of the SNTTEXT string or a tilde and match until the end of the string or a tilde. The following sentence output values will be assigned to the case corresponding to SNTTEXT data 1-4:

cnf = 1 incmax = 99299299.22 incmin = 99899899.88

4.5.2.6 Tilde Match

The tilde match rule transforms the user defined plaintext key into a regular expression pattern which will attempt to match the given plaintext key **from the start of the SNTTEXT string to the end of the string or a tilde** and if a match is found assign the provided output values.

Rule Type

:re:tilde-m:

Constraints

- 1) Key pattern must match at start of the SNTTEXT data
- 2) Key pattern must match to the end of the SNTTEXT data or be proceeded by a tilde

Usage

SNTTEXT Data 1 = “life in prison”

SNTTEXT Data 2 = “no court fees~life in prison”

SNTTEXT Data 3 = “life in prison~no court fees”

SNTTEXT Data 4 = “no probation~life in prison~no court fees”

SNTTEXT Data 5 = “sentenced to life”

SNTTEXT Data 6 = “no court fees~sentenced to life in prison”

SNTTEXT Data 7 = “sentenced to life in prison~no court fees”

SNTTEXT Data 8 = “no probation~sentenced to life in prison~no court fees”

Rule Crosswalk Format

Ruletype	Key	cnf	Incmax	incmin
:re:tilde-m:	life in prison	1	99299299.22	99899899.88

Machine Format

:re:tilde-m:life in prison 1|99299299.22|99899899.88 1

Result

The above rule will match on SNTTEXT data 1 and SNTTEXT data 3, because the plaintext key matches “life in prison” and “life in prison~” from the start of the string to the end of the string or a tilde. The above tilde rule does match “life in prison” on SNTTEXT data 2 and SNTTEXT data 4 because “life in prison” does not start at the beginning of the string or a tilde, and does not match on SNTTEXT data 5 – SNTTEXT data 8 because “life in prison” does not appear at the beginning of the string and match until the end of the string or a tilde. The following output values will be assigned to the case corresponding to SNTTEXT data 1 and SNTTEXT data 3:

cnf = 1 incmax = 99299299.22 incmin = 99899899.88

Variations

Name	Rule Type	Variation Info
Tilde Match Rule	:re:tilde-match:	Alternative rule type for Tilde Match Rule

4.5.2.7 Tilde Begin

The tilde begin rule transforms the user defined plaintext key pattern into a regular expression pattern which will attempt to match the given plaintext key **that begins at the start of the SNTTEXT data or is preceded by a tilde** and if a match is found assign the provided output values.

Rule Type

:re:tilde-b:

Constraints

- 1) Key pattern must start at the beginning of the string or be preceded by a tilde

Usage

SNTTEXT Data 1 = “life in prison”

SNTTEXT Data 2 = “no court fees~life in prison”

SNTTEXT Data 3 = “life in prison~no court fees”

SNTTEXT Data 4 = “no probation~life in prison~no court fees”

SNTTEXT Data 5 = “sentenced to life”

SNTTEXT Data 6 = “no court fees~sentenced to life in prison”

SNTTEXT Data 7 = “sentenced to life in prison~no court fees”

SNTTEXT Data 8 = “no probation~sentenced to life in prison~no court fees”

Rule Crosswalk Format

Ruletype	Key	cnf	Incmax	incmin
:re:tilde-b:	life in prison	1	99299299.22	99899899.88

Machine Format

```
:re:tilde-b:life in prison      1|99299299.22|99899899.88      1
```

Result

The above rule will match on SNTTEXT data 2, 3 and 4 because the plaintext key matches “life in prison” at the start of the string or is preceded by a tilde. The above tilde rule does match “life in prison” on the remaining keys because wither “life in prison” is not matched (SNTTEXT 5) or violates the Tilde Begin constraints. The following output values will be assigned to the case corresponding to the sentence output values for the SNTTEXT output variable associated with SNTTEXT data 2, 3 and 4:

```
cnf = 1          incmax = 99299299.22          incmin = 99899899.88
```

Variations

Name	Rule Type	Variation Info
Tilde Begin Rule	:re:tilde-begin:	Alternative rule type for Tilde Begin Rule

4.5.2.8 Math Search

Operates the same as the Search rule (see 4.5.2.1 above) but allows formulas to be assigned to the sentencing output values. The digit groups extracted from the SNTTEXT data will be used in the output formulas. The digit groups are referenced by #n where n = the index of # in the rule key pattern. If there are multiple occurrences of the key pattern in the SNTTEXT data, the first occurrence of the key pattern will be used for digit group capture. To allow digit capture on all occurrences of the key pattern use the Math Find All rule (see 4.5.2.12 below).

Rule Type

```
:re:math-s:
```

Usage

SNTTEXT Data1 = “Sentenced to 5 days 10 years to 20 years in prison”

SNTTEXT Data 2 = “Sentenced to days 10 years to 20 years in prison”

Rule Crosswalk Format

Ruletype	Key	Cnf	incmax	Incmin
:re:math-s:	# days # years to # years	1	#3	#1+#2(365)

Machine Format

```
:re:math-s:# days # years to # years      1|#3|#1+#2(365)
```

Result

The above rule will not match on SNTTEXT data 2 because a number does not precede “days” but it does find a match on “5 days 10 years to 20 years” in SNTTEXT data 1 and codes the sentencing output variables by placing the captured digit groups into the value formulas:

```
cnf = 1
incmax = #3 = 20
incmin = #1 + #2(365) = 5 + 10(365) = 3655
```

4.5.2.9 Math Match

Operates the same as the Match rule (see 4.5.2.2 above) but allows formulas to be assigned to the sentencing output values. The digit groups extracted from the SNTTEXT data will be used in the output formulas. The digit groups are referenced by #n where n = the index of # in the rule key pattern.

Constraints

- 1) Key pattern must match to at the start of the SNTTEXT data

Usage

SNTTEXT Data 1 = “5 days 10 years to 20 years in prison”

SNTTEXT Data 2 = “Sentenced to 5 days 10 years to 20 years in prison”

Rule Crosswalk Format

Ruletype	Key	cnf	incmax	incmin
:re:math-m:	# days # years to # years	1	#3	#1+#2(365)

Machine Format

```
:re:math-m:# days # years to # years    1|#3|#1+#2(365)
```

Result

The above rule will find “5 days 10 years to 20 years” in SNTTEXT data 1 because the key pattern matches and it starts at the beginning of the string but does not find a match on SNTTEXT data 2 because the key pattern is not found at the beginning of SNTTEXT data 2. The captured digit groups from the SNTTEXT data 1 string will be substituted into the output value digit group placeholders. The output formulas will be computed and assigned to the sentencing output values:

```
cnf = 1
incmax = #3 = 20
incmin = #1 + #2(365) = 5 + 10(365) = 3655
```

4.5.2.10 Math Exact

Operates the same as the Exact rule (see 4.5.2.3 above) but allows formulas to be assigned to the output values. The values captured in the SNTTEXT Data will be used in the output formulas. The captured key values are referenced by #n where n = the index of # in the rule key.

Rule Type

:re:math-e:

Constraints

- 1) Key pattern must match from the beginning to the end of the SNTTEXT data

Usage

SNTTEXT Data 1 = “5 days 10 years to 20 years”

SNTTEXT Data 2 = “5 days 10 years to 20 years in prison”

Rule Crosswalk Format

Ruletype	Key	Cnf	incmax	Incmin
:re:math-e:	# days # years to # years	1	#3	#1+#2(365)

Machine Format

:re:math-e:# days # years to # years 1|#3|#1+#2(365)

Result

The above rule will find “5 days 10 years to 20 years” in the SNTTEXT data 1 because the key matches from the beginning of the string to the end of the string but does not find a match on key data 2 because it ends with “in prison”. The output values associated with SNTTEXT data 1 will be coded as:

cnf = 1

incmax = #3 = 20

incmin = #1 + #2(365) = 5 + 10(365) = 3655

4.5.2.11 Math End

Operates the same as the End rule (see 4.5.2.4 above) but allows formulas to be assigned to the sentence output values. The values captured in the SNTTEXT Data will be used in the output formulas. The captured key values are referenced by #n where n = the index of # in the rule key.

Rule Type

:re:math-end:

Constraints

- 1) Key pattern must match to the end of the SNTTEXT data

Usage

SNTTEXT Data 1 = “~5 days 10 years to 20 years~”

SNTTEXT Data 2 = “Sentenced to 5 days 10 years to 20 years”

Rule Crosswalk Format

Ruletype	Key	Cnf	incmax	Incmin
:re:math-end:	# days # years to # years	1	#3	#1+#2(365)

Machine Format

:re:math-end:# days # years to # years 1|#3|#1+#2(365)

Result

The above rule will match on “5 days 10 years to 20 years” in the SNTTEXT data 2 because the plaintext key pattern matches from the start of the beginning to the end of the SNTTEXT string string to the end of the string. The plaintext key pattern does not match on SNTTEXT data 2 because the key pattern is not found within the tilde rule constraints. does not find a match on key data 2 because it ends with “in prison”. The output values associated with key data 1 will be coded as:

cnf = 1

incmax = #3 = 20

incmin = #1 + #2(365) = 5 + 10(365) = 3655

4.5.2.12 Math Find All

Operates the same as the Math Search rule (see 4.5.2.8 above), but instead of only capturing the first occurrence of the key pattern in the SNTTEXT data the Find All rule matches all occurrences, calculates the sentence value output formula for each of the matched patterns and then returns the highest value found. This provides the ability for selecting the highest matched key pattern in the SNTTEXTX data.

Rule Type

:re:math-f:

Usage

Key Data 1 = “sentenced to 5 years to 10 years, 10 years to 20 years, 5 years to 50 years”

Rule Crosswalk Format

Ruletype	Key	Cnf	incmax	Incmin
:re:math-f:	# years to # years	1	#2*365	#1*365

Machine Format

```
:re:math-f:# years to # years    1|#2*365|#1*365
```

Result

The above rule will match all occurrences of the key pattern “# years to # years” in the SNTTEXT data. This will result in finding matches on “5 years to 10 years”, “10 years to 20 years”, and “20 years to 50 years”. Each of these will be matched occurrences will be inputted into each of the output formulas. Each output formula will select the highest possible output value and assign that to the sentence variable.

cnf = 1

incmax = 18,250 (selected from c below because it evaluated to the highest amount)

a) $\#2*365 = 10 *365 = 3,650$

b) $\#2*365 = 20*365 = 7,300$

c) $\#2*365= 50*365 = 18,250$

incmin = 3,650 (selected from b below because it evaluated to the highest amount)

a) $\#1*365 = 5 *365 = 1,825$

b) $\#1*365 = 10*365 = 3,650$

c) $\#1*365= 5*365 = 1,825$

4.5.2.13 Math Tilde

Operates the same as the Tilde rule (see 4.5.2.5 above) but allows formulas to be assigned to the sentencing output values. The digit groups are extracted from the SNTTEXT data and can be referenced in the sentencing output value formulas. The digit groups are referenced by #n where n = the index of # in the rule key pattern.

Rule Type

```
:re:math-t:
```

Constraints

- 1) Key pattern must match at start of the SNTTEXT data or by preceded by a tilde
- 2) Key pattern must match to the end of the SNTTEXT data or by proceeded by a tilde

Usage

SNTTEXT Data 1 = “~5 days 10 years to 20 years~”

SNTTEXT Data 2 = “Sentenced to 5 days 10 years to 20 years”

Rule Crosswalk Format

Ruletype	Key	Cnf	incmax	Incmin
:re:math-t:	# days # years to # years	1	#3	#1+#2(365)

Machine Format

```
:re:math-t:# days # years to # years      1|#3|#1+#2(365)
```

Result

The above rule will not match SNTTEXT data 2 because it violates the Tilde constraints, the key pattern does not start at the beginning of the string and is not preceded by a tilde. However, the above rule does match “5 days 10 years to 20 years” in SNTTEXT data 1 and since the match does not violate the Tilde constraints the digit groups will be substituted into the sentence output value formulas. The sentence output values will be coded as:

```
cnf = 1
incmax = #3 = 20
incmin = #1 + #2(365) = 5 + 10(365) = 3655
```

4.5.2.14 Math Tilde Match

Operates the same as the Tilde Match rule (see 4.5.2.6 above) but allows mathematical formulas to be used in the sentencing output values. The digit groups are extracted from the SNTTEXT data and can be referenced in the sentencing output value formulas. The digit groups are referenced by #n where n = the index of # in the rule key pattern.

Rule Type

```
:re:math-t-m:
```

Constraints

- 1) Key pattern must match at start of the SNTTEXT data
- 2) Key pattern must match to the end of the SNTTEXT data or be proceeded by a tilde

Usage

SNTTEXT Data 1 = “~5 days 10 years to 20 years~”

SNTTEXT Data 2 = “5 days 10 years to 20 years~”

SNTTEXT Data 3 = “5 days 10 years to 20 years”

Rule Crosswalk Format

Ruletype	Key	Cnf	incmax	Incmin
:re:math-t-m:	# days # years to # years	1	#3	#1+#2(365)

Machine Format

```
:re:math-t-m:# days # years to # years 1|#3|#1+#2(365)
```

Result

The above rule will not match SNTTEXT data 1 because it violates the Tilde Match constraint, the key pattern does not start at the beginning of the string. However, the above rule does match “5 days 10 years to 20 years” in SNTTEXT data 1 and SNTTEXT data 2 without violating the Tilde Match constraints so the digit groups will be substituted into the sentence output value formulas. The sentence output values will be coded as:

```
cnf = 1
incmax = #3 = 20
incmin = #1 + #2(365) = 5 + 10(365) = 3655
```

4.5.2.15 Math Tilde Begin

Operates the same as the Tilde Match rule (see 4.5.2.6 above) but allows mathematical formulas to be used in the sentencing output values. The digit groups are extracted from the SNTTEXT data and can be referenced in the sentencing output value formulas. The digit groups are referenced by #n where n = the index of # in the rule key pattern.

Rule Type

```
:re:math-t-b:
```

Constraints

- 1) Key pattern must start at the beginning of the SNTTEXT data or be preceded by a tilde

Usage

SNTTEXT Data 1 = “Sentenced to 5 days 10 years to 20 years”

SNTTEXT Data 2 = “~5 days 10 years to 20 years”

SNTTEXT Data 3 = “5 days 10 years to 20 years in prison”

Rule Crosswalk Format

Ruletype	Key	Cnf	incmax	Incmin
:re:math-t-b:	# days # years to # years	1	#3	#1+#2(365)

Machine Format

```
:re:math-t-b:# days # years to # years 1|#3|#1+#2(365)
```

Result

The above rule will not match SNTTEXT data 1 because it violates the Tilde Match constraint, the key pattern does not start at the beginning of the string or is preceded by a tilde. However, the above rule does match “5 days 10 years to 20 years” in SNTTEXT data 1 and SNTTEXT data 2 without violating the Tilde Match constraints so the digit groups will be substituted into the sentence output value formulas. The sentence output values will be coded as:

```
nf = 1
incmax = #3 = 20
incmin = #1 + #2(365) = 5 + 10(365) = 3655
```

4.5.2.16 Regex

The regex rule takes the user defined regular expression pattern (accepting all python regular expression rules <http://docs.python.org/2.7/library/re.html>) and attempts to match this pattern **anywhere** within the key data. If a match is found assign the provided sentencing output values. The rule key does not support plaintext keys; do not use a regex rule if a plaintext key is desired.

Rule Type

:re:regex:

Usage

SNTTEXT Data 1 = “Sentenced to life in prison”

SNTTEXT Data 2 = “Sentenced to lif in prison”

Rule Crosswalk Format

Ruletype	Key	cnf	Incmax	incmin
:re:regex:	life in prison lif in prison	1	99299299.22	99899899.88

Machine Format

```
:re:search:life 1|99299299.22|99899899.88 1
```

Result

The above rule will match the string “life in prison” or the string “lif in prison” in the SNTTEXT data 1 and 2. The bar character represent a OR assignment, instructing the pattern to match the first regular expression OR the second regular expression. The following sentence output variables will be coded as:

```
cnf = 1          incmax = 99299299.22          incmin = 99899899.88
```

Variations

Name	Machine Format	Variation Info
Regex Search Rule	:re:regex-s:	Works exactly the same as Regex rule
Regex Match Rule	:re:regex-m:	Same as a Match rule (4.5.2.2) but key pattern is a regular expression instead of a plaintext key
Regex Exact Rule	:re:regex-e:	Same as an Exact rule (4.5.2.3) but key pattern is a regular expression instead of a plaintext key
Regex End Rule	:re:regex-end:	Same as a End rule (4.5.2.4) but key pattern is a regular expression instead of a plaintext key
Regex Tilde	:re:regex-t:	Same as a Tilde rule (4.5.2.5) but key pattern is a regular expression instead of a plaintext key
Regex Tilde Match	:re:regex-t-m:	Same as a Tilde Match rule (4.5.2.6) but key pattern is a regular expression instead of a plaintext key
Regex Tilde Begin	:re:regex-t-b:	Same as a Tilde Begin rule (4.5.2.7) but key pattern is a regular expression instead of a plaintext key
Regex Math Search	:re:regex-math-s:	Same as a Math Search rule (4.5.2.8) but key pattern is a regular expression instead of a plaintext key
Regex Math Match	:re:regex-math-m:	Same as a Math Match rule (4.5.2.9) but key pattern is a regular expression instead of a plaintext key
Regex Math Exact	:re:regex-math-e:	Same as a Math Exact rule (4.5.2.10) but key pattern is a regular expression instead of a plaintext key
Regex Math End	:re:regex-math-end:	Same as a Math End rule (4.5.2.11) but key pattern is a regular expression instead of a plaintext key
Regex Math Find All	:re:regex-math-f:	Same as a Math Find All rule (4.5.2.12) but key pattern is a regular expression instead of a plaintext key
Regex Math Tilde	:re:regex-math-t:	Same as a Math Tilde rule (4.5.2.13) but key pattern is a regular expression instead of a plaintext key
Regex Math Tilde Match	:re:regex-math-t-m:	Same as a Math Tilde Match rule (4.5.2.14) but key pattern is a regular expression instead of a plaintext key
Regex Math Tilde Begin	:re:regex-math-t-b:	Same as a Math Tilde Begin rule (4.5.2.15) but key pattern is a regular expression instead of a plaintext key

5 – System Output

5.1 Relational Databases

The BJS CCHRRD Conversion system creates two MySQL databases during a projects conversion life. Both of the databases are created and initialized during the data staging phase and then populated with data throughout the remaining conversion phases. During the data staging phase the following actions are performed to the standardized and recidivism databases:

1. Databases are created in the MySQL server. The standardized and recidivism databases will be named based on the db_stand and db_recid configuration settings (default is *projectname* and *projectname_recid*). The MySQL server is connected to using the db project configuration settings. See section 2.2 for more details on the setting the initial MySQL server configuration and section 2.2.3 for more details on the project configuration file.
2. Lookup tables are created in the two databases based on the business logic database metadata. See section 4.1.2 for more information on the business logic database metadata.
3. The subject files and the NLETS intermediate arrest, court-sentencing, demographic, and supervision files are loaded into tables within the standardized database. The subject table is the primary entity in the database, it contains one row for each subject with the primary key being the subject's casenum. All the records within the segment tables are linked back to the subject using a foreign key constraint on the casenum field. The arrest, court-sentencing, demographic, and supervision segment data are loaded into archive segment tables during the stage phase. These archive tables contain all the original data in string format.

5.2 SPSS Flat Files

5.2.1 Standardized Database Extract Files

- ▶ SPSS standardized relational database extract files are created from the segment tables after the process phase is complete (see section 3.3.3 for more details on the process phase). These files are saved to: *projectdir/output/review/set/segment/region/round*

5.2.2 Recidivism Database Extract Files

- ▶ SPSS relational database extract files are created from the arrest and sentence segment tables after the Recidivism Database Load phase is complete (see section 3.3.5 for more details on the Recidivism Database Load phase). These files are named after the segment name. n

addition all the records that were removed from these segments based on the recidivism coding rules are saved to SPSS files. These files are named *segment_del_records*. All these files are saved to *projectdir/output/transform*

5.2.3 Recidivism Flat Files

- The SPSS Criminal History Flat, Criminal History Summary, and Demographic Flat files are created after the Recidivism Flat File Load phase is complete (see section 3.3.6 for more details on this phase). These files are saved to their respective directories in *projectdir/output/recid*

5.3 Quality Control Report Files

5.3.1 Conversion Reports

Conversion reports are created based on the conversion logs generated during the pre-process and process conversion phases. The logs contain information from any potential or actual conversion rule error that occurs within the rule controllers during data processing. The pre-process report files are saved to *projectdir/output/report/pre-process* and the process rule files are saved to *projectdir/output/review/set/segment/round*.

While a variety of reports are created the most important reports that are created is the offense conversion reports. These reports list all arrest and sentence records that were not supported by the offense coding rules and the reports will only be created if errors are detected. There are two possible reports for the two segments; a report detailing all the WVFBI offense conversion errors, *report_ancic_wvfbi* for the arrest segment and *report_cncic_wvfbi* for the court-sentence segment, and a report detailing all the state offense conversion errors, *report_ancic* and *report_cncic*.

5.3.1.1 State Offense Conversion Reports

5.3.1.2 WVFBI Offense Conversion Reports

The WVFBI offense conversion report contains the following information:

Column Name	Description
segment	Provides the id corresponding to the segment the error occurred in
variable	The variable rule manager that the error occurred in
phase	Coding rule phase (2 = process)
id	The unique ID index of the record that failed to be processed by the offense conversion rules
Convert State	The region of the record (record value for a/cstate field)

Convert Key	The key value that the ruleset was unable to handle
Error Message	The reason the offense conversion failed
Convert Time	A timestamp for the conversion run

The WVFBI offense conversion reports will list all the records with a key value (a/cchglitx field) that do not have a ruleset cannot handle. Remember, the ruleset contains a list of rules with each rule being able to handle a certain key(s). So if the ruleset does not have a rule that can handle the key associated with a record then the record will be logged as failing offense conversion.

How to use the report

All of the key values from the WVFBI offense conversion report must be added to the WVFBI offense rulesets. The key must be added to the WVFBI ruleset corresponding to the state value in the report, and the output values must be filled. After the rules have been updated and the process phase has been run again, check to see if a WVFBI offense report still exists. If it does not then rule updates were successful, otherwise more updates need to be made.

Example

If a row has a rapstate = 'WVFBI' then it will be coded according to the wvfbi rule controller and rule sets. If the record is not a federal record (afed=2) than it will be coded according to the Alaska ruleset (astate=2=Alaska), otherwise it would have been coded using the FED ruleset (afed=1). Since the key value of "murder second degree" is not present in the ruleset it will be added to the WVFBI offense conversion report. Upon seeing this report the user should take the "murder second degree" string and add it to the coding rules with the output values of 999, 10, 9, 9, 9, 99, 9.

Row Value

ID	casenum	afed	rapstate	astate	achglitx
100002	Adom	2	WVFBI	2	murder second degree

Example Alaska Ruleset

:vars: ancic|achg|ainc|adom|aweap|acd|apg

:lookup: ncic|chg|inc|dom|weap|cdv|pg

:keys: achglitx

Murder 999|10|9|9|9|99|9 1

Murder 2nd degree 999|10|9|9|9|99|9 1

Example Federal Record Offense Conversion Report (report_ancic_wvfbi)

segment	variable	phase	id	state	key	error	Time
2	ancic	2	100002	AK	Murder second degree	key not in crosswalk	...

5.4 Comparison Reports

Comparison reports provide information on how a process region file has changed from one region to another. For instance if a project has run the Process command two times on the Arizona region the difference report will saved to *projectdir/output/review/set/arrest/round2/report_diff.xls*. The process difference report will consist of an **id** column which contains the id of a record where a change was detected, the **region** integer value, the **rapstate**, the **variable name**, and the value from the previous dataset (**orig_value**) and the new value (**new_value**).

Example of process difference report

id	region	rapstate	var_name	orig_value	new_value
28895	17.0	WVFB	ancic	1313.0	1399.0
28895	17.0	WVFB	achg	130.0	135.0
28897	17.0	WVFB	ancic	5707.0	2399.0

Comparison reports are also available for the summary file. The summary difference report will consist of a column for the unique **casenum**, the variable name of the where a change was detected (**var_name**), the value from the previous summary file (**orig_value**), and the new value in the current summary file (**new_value**).

Example of summary difference report

casenum	var_name	orig_value	new_value
28895	prarr	0	1
28895	pradj	0	1
28897	Rear	0	1

5.5 Referential Integrity Report

The referential integrity report lists all occurrences where a rule output value is not present in the corresponding lookup table for the output variable. The report contains the following information:

Column Name	Description
segment	Provides the id corresponding to the segment the error occurred in
variable	The output variable that failed a referential integrity check
phase	Coding rule phase
bad_value	Rule output value that does not have a corresponding value in the lookup table associated with the ruleset output variable
table_name	The name of the lookup table that the output value must conform to
allowed_values	A list of the values within the lookup table. The output value must match one of these values
rule_loc	The path to the ruleset containing the problematic rule
error_message	Message describing the referential integrity error

Example:

If a ruleset has an output variable of adom, with adom only allowing output values of 1,2, and 9, and one of the ruleset rules has an output value of 12 a referential integrity error will be saved to the report because 12 is not within the allowed values of adom.

Example Ruleset

```
:vars:  adom
:lookup:  dom
:keys:  achglitx
Arrested for domestic violence      12      1
```

Example Referential Integrity Report

segment	variable	phase	bad_value	table_name	allowed_values	rule_loc	error
2	adom	3	12	dom	[1,2,9]	<i>Path to ruleset file</i>	Value not present in lookup

5.6 Regular Expression Validation Report

This report lists all occurrences where a regular expression rule does not pass a rule validation check. The report contains the following information:

Column Name	Description
Message	Indicates the severity and reason of validation failure
Filename	Provides the path to the ruleset containing the problematic rules
Regex_pattern_1	The regular expression pattern of the potentially problematic rule (earlier priority)

Regex_result_1	The output value associated with the potentially problematic rule (earlier priority)
Regex_priority_1	The name of the lookup table that the output value must conform to
Regex_pattern_2	A list of the values within the lookup table. The output value must match one of these values
Regex_result_2	The path to the ruleset containing the problematic rule
Regex_priority_2	Message describing the referential integrity error

There are three levels of failure: warning, potential error, and error. The warning is the least severe failure and simply indicates that the rule does not follow best practices but will not result in conversion or system errors. In the below example we are attempting to code the incmax field by searching the snttextx field using two regular expression rules. The first rule looks for “life” and if it finds a match codes this to 99299299.22. The second string looks for “sentenced to life”, however the second rule is not needed because the first rule will match all instances of life resulting in the second rule never being checked. The validation check will discover and then log the details of this unnecessary rule.

Example Ruleset

```
:vars:  incmax
:key:   snttextx
:re:match:life      99299299.22      1
:re:match:sentenced to life  99299299.22  2
:re:match:life in prison 99797299.77  3
:re:match:(\d+) years      #2*365  4
```

Example Referential Integrity Report

message	file	pattern_1	result_1	priority1	pattern2	result2	Priority2
Warning: redundant rule	<i>Path to file</i>	life	99299299.22	1	sentenced to life	99299299.22	2
Potential Error: Inconsistent Results	<i>Path to file</i>	life	99299299.22	1	life in prison	99799799.77	3
error: Syntax Error, formula number index out of range	<i>Path to file</i>	(\d+) years	#2*365	4	<i>timestamp</i>		

A potential error indicates that there are two conflicting regular expression rules within a crosswalk which may result in improper coding of an output value. In the above example the first rule, a search for “life”, and the third rule, search for “life in prison”, have different output values. This is the same as a redundant rule except for the output value differing between the two fields, indicating that one of the values is incorrect. If the rule with a higher priority has the incorrect output value, this will result in an incorrect

coding. In the above example, rule 1 has a higher priority with the correct output value. Therefore, the incmax field will be correctly coded by rule 1 even though the error exists in the rules.

The third type of validation failures are errors, which will result in a rule error during processing resulting in a null value being assigned to the output variable(s). In the above example rule 4 has syntax error where the output value has an invalid formula. The search string “(\d+) years” is only attempting to match with one number group, but in the output value the formula is referencing the second number group. Since this second group does not exist the rule will raise a RuleError during processing and assign a “null” to the incmax field.

5.7 Fatal Error Reports

Under ideal conditions a fatal error should not occur in the system. However, in an effort to stop the program from unexpectedly terminating all system errors are caught and logged by an exception handler. If the unexpected error occurs during a process coding rule conversion it will be added to a fatal error report.

Column Name	Description
Segment	Provides the id corresponding to the segment the error occurred in
Variable	The variable rule manager that the error occurred in
Phase	Coding rule phase
error message	The system error that occurred (python exception string)
Time	The time the error occurred

If a fatal error is detected advanced system diagnosis is required. To see a full stack exception trace the user can set the halt_on_error configuration setting to True. Additionally, if the fatal error was not a coding rule error but a general system error the exception will be logged to the command log error directory (*projectdir/tmp/command/error/timestamp*) or the full stack exception can be raised by running the command again with the verbose command line parameter (-v).

Appendix

Appendix A: Standardized Database Schema

A.1 Subject Segment

Table

Subject (id, state, moob, daob, yrob, morl, darl, yrri)

Domains

Variable Name	Format	Length	Constraint
Id	INT	11	PK, NOT NULL
State	INT	11	
Moob	INT	11	
Daob	INT	11	
Yrob	INT	11	
Morl	INT	11	
Darl	INT	11	
Yrri	INT	11	

Constraints

PRIMARY KEY id

Notes

Subject segment is the primary entity for the standardized relation database. The id value is set to subject casenum assuming that the casenum will be able to uniquely identify each record. The total number of records should correspond to the total number of subjects in the study with one record per subject. This ensures entity integrity in Subject (no duplicate ids i.e. casenums). Since all other entities are linked to Subject id field via a foreign key constraint on the casenum field we must have at a minimum a id (casenum) for each subject inserted into the Subject table before we can begin inserting data into the other tables. This enforces referential integrity on casenum in the database (in order for a record to be added to a segment table the casenum of the record must be present in the id field within the Subject table).

A.2 Arrest Segment

A.2.1 Standardized Table

Arrest (id, casenum, rectype, nonarr, astate, asource, afed, cycnum, aseqnum, arrayr, armo, arrda, arname, aori_invalid, aori, offyr, offmo, offda, atype, achglit, achgdet, achgncic, aoffcode, ancic, achg, achg16, achgsvr, ainc, adom, aweap, acdv, apg, acnt, adisptextx, adsp)

Variable Name	Format	Length	Constraint
id	INT	11	PK, FK(Arrestx.id), NOT NULL
casenum	INT	11	FK(Subject.id)
rectype	INT	11	FK(Segment.id)
nonarr	INT	11	FK(Admrec.id)
astate	INT	11	FK(Region.id)
asource	INT	11	FK(Source.id)
afed	INT	11	FK(Fed.id)
cycnum	INT	11	
aseqnum	INT	11	
arrayr	INT	11	
armo	INT	11	
arrda	INT	11	
arname	VARCHAR	60	
aori_invalid	TINYINT	1	
aori	VARCHAR	9	
offyr	INT	11	
offmo	INT	11	
offda	INT	11	
atype	INT	11	FK(Type.id)
achglit	VARCHAR	128	
achgdet	VARCHAR	87	
achgncic	VARCHAR	4	
aoffcode	VARCHAR	20	
ancic	INT	11	FK(Ncic.id)
achg	INT	11	FK(Chg.id)
achg16	INT	11	FK(Chg16.id)
achgsvr	INT	11	FK(Severity.id)
ainc	INT	11	FK(Inc.id)
adom	INT	11	FK(Dom.id)
aweap	INT	11	FK(Weap.id)
acdv	INT	11	FK(Cdv.id)
apg	INT	11	FK(Pg.id)
acnt	INT	11	
adisptextx	VARCHAR	83	
adsp	INT	11	FK(Dsp.id)

Key Constraints

PRIMARY KEY id

FOREIGN KEY id
REFERENCES Arrestx (id)

FOREIGN KEY casenum
REFERENCES Subject (id)

FOREIGN KEY rectype
REFERENCES Segment (id)

FOREIGN KEY nonarr
REFERENCES Admrec (id)

FOREIGN KEY astate
REFERENCES Region (id)

FOREIGN KEY asource
REFERENCES Source (id)

FOREIGN KEY afed
REFERENCES Fed (id)

FOREIGN KEY atype
REFERENCES Type (id)

FOREIGN KEY ancic
REFERENCES Ncic (id)

FOREIGN KEY achg
REFERENCES Chg (id)

FOREIGN KEY achg16
REFERENCES Chg16 (id)

FOREIGN KEY achgsvr
REFERENCES Severity (id)

FOREIGN KEY ainc
REFERENCES Inc (id)

FOREIGN KEY adom
REFERENCES Dom (id)

FOREIGN KEY aweap
REFERENCES Weap (id)

FOREIGN KEY apg
REFERENCES Pg (id)

FOREIGN KEY acdv
REFERENCES Cdv (id)

FOREIGN KEY adsp
REFERENCES Dsp (id)

A.2.2 Archive Table

Arrestx (id, casenum, casenumx, rectype, arrstatex, astate, rapstatex, asource, ynfederalx, afed, cycnumx, cycnum, aseqnumx, arrdatex, arname, arnamex, aori_invalid, aorix, aori, offdatex, arrtypex, astatnumx, achglitx, achgdetx, achgncicx, aoffcodex, achgsvr, atrnumx, aincx, arrcrfx, arrcefx, achgcntx, aactlix, adispdatex, adisptextx, adisptypex, eventdate1x)

Variable Name	Format	Length	Constraint
id	INTEGER	10	PK, NOT NULL, AUTO INC
casenum	INTEGER	10	FK(Subject.id)
casenumx	TEXT		
rectypex	TEXT		
arrstatex	TEXT		
astate	INTEGER	2	FK(Region.id)
rapstatex	TEXT		
asource	INT	1	FK(Source.id)
ynfederalx	TEXT		
afed	INT	1	FK(Fed.id)
cycnumx	TEXT		
cycnum	INTEGER	3	
aseqnumx	VARCHAR		
aseqnum	INTEGER	3	
arrdatex	TEXT		
arname	VARCHAR	60	
arnamex	TEXT		
aori_invalid	TINYINT	1	
aorix	TEXT		
aori	VARCHAR	9	
offdatex	TEXT		
arrtypex	TEXT		
astatnumx	TEXT		
achglitx	TEXT		
achgdetx	TEXT		
achgncicx	TEXT		
aoffcodex	TEXT		
achgsvr	TEXT		
atrnumx	TEXT		
aincx	TEXT		
arrcrfx	TEXT		
arrcefx	TEXT		
achgcntx	TEXT		
aactlix	TEXT		
adispdatex	TEXT		
adisptextx	TEXT		
adisptypex	TEXT		
eventdate1x	TEXT		

Foreign Key Constraints

```
PRIMARY KEY id

FOREIGN KEY casenum
    REFERENCES Subject(id)
FOREIGN KEY astate
    REFERENCES Region(id)
FOREIGN KEY asource
    REFERENCES Source(id)
FOREIGN KEY afed
    REFERENCES Fed(id)
```

Notes

Id will be automatically assigned on data load giving each arrest record a unique identifier.

A.3 Court-Sentence Segment

A.3.1 Standardized Table

Sentence (id, casenum, rectype, cstate, csource, cfed, cycnum, cseqnum, cchglit, cchgdet, cchgncic, coffcode, cncic, cchg, cchg16, cchgsvr, cinc, ccnt, cdispyr, cdispmo, cdispda, cdisptextx, cdsp, sntdatex, sntyr, sntmo, sntda, cnf, trt, oth, cmsrv, incmin, incmax, susp, suspmin, suspmax, suspfine, prb, costs, fineamt, rst, cori, cori_invalid, cdom, cpg, cweap, cdv, noncrt, pjp)

Variable Name	Format	Length	Constraint
<u>id</u>	INT	11	PK, FK(Sentencex.id), NOT NULL
casenum	INT	11	FK(Subject.id)
rectype	INT	11	FK(Segment.id)
cstate	INT	11	FK(Region.id)
csource	INT	11	FK(Fed.id)
cfed	INT	11	
cycnum	INT	11	
cseqnum	INT	11	
cchglit	VARCHAR	128	
cchgdet	VARCHAR	256	
cchgncic	VARCHAR	4	
coffcode	VARCHAR	20	
cncic	INT	11	FK(Ncic.id)
cchg	INT	11	FK(Chg.id)
cchg16	INT	11	FK(Chg16.id)
cchgsvr	INT	11	FK(Severity.id)
cinc	INT	11	FK(Inc.id)
ccnt	INT	11	
cdispyr	INT	11	
cdispmo	INT	11	
cdispda	INT	11	
cdisptextx	VARCHAR	512	
cdsp	INT	11	FK(Dsp.id)
sntdatex	VARCHAR	11	
sntyr	INT	11	
sntmo	INT	11	
sntda	INT	11	
cnf	INT	11	FK(Cnf.id)
trt	INT	11	FK(Trt.id)
oth	INT	11	FK(Oth.id)
cmsrv	DECIMAL	13,2	
incmin	DECIMAL	13,2	
incmax	DECIMAL	13,2	
suspmax	DECIMAL	13,2	
suspmin	DECIMAL	13,2	
suspfine	DECIMAL	13,2	
prb	DECIMAL	13,2	
costs	DECIMAL	13,2	
fineamt	DECIMAL	13,2	

Variable Name	Format	Length	Constraint
rst	DECIMAL	13,2	
cori	VARCHAR	9	
cori_invalid	TINYINT	1	
cdom	INT	11	FK(Dom.id)
cpg	INT	11	FK(Pg.id)
cweap	INT	11	FK(Weap.id)
ccdvd	INT	11	FK(Cdv.id)
noncrt	INT	11	FK(Admrec.id)
pjp	INT	11	FK(Pjp.id)

Constraints

PRIMARY KEY id

FOREIGN KEY id

REFERENCES Sentencex (id)

FOREIGN KEY casenum

REFERENCES Subject (casenum)

FOREIGN KEY rectype

REFERENCES Segment (id)

FOREIGN KEY cstate

REFERENCES Region (id)

FOREIGN KEY csource

REFERENCES Source (id)

FOREIGN KEY cfed

REFERENCES Fed (id)

FOREIGN KEY cncic

REFERENCES Ncic (id)

FOREIGN KEY cchg

REFERENCES Chg (id)

FOREIGN KEY cchg16

REFERENCES Chg16 (id)

FOREIGN KEY cchgsvr

REFERENCES Severity (id)

FOREIGN KEY cinc

REFERENCES Inc (id)

FOREIGN KEY cdsp

REFERENCES Dsp (id)

FOREIGN KEY cnf

REFERENCES Cnf (id)

FOREIGN KEY trt

REFERENCES Trt (id)

FOREIGN KEY oth

REFERENCES Oth (id)

FOREIGN KEY cdom

REFERENCES Dom (id)

FOREIGN KEY cpg

REFERENCES Pg (id)

```

FOREIGN KEY cweap
    REFERENCES Weap (id)
FOREIGN KEY ccdv
    REFERENCES Cdv (id)
FOREIGN KEY noncrt
    REFERENCES Admrec (id)
FOREIGN KEY pjp
    REFERENCES Pjp (id)

```

A.3.2 Archive Table

Sentencex (id, casenum, casenumx, rectype, crtstatex, cstate, rapstatex, csource, ynfederalx, cfed, cycnumx, cycnum, cseqnumx, cseqnum, crtcmtx, crtnamex, cstatnumx, cchglitx, cchgdetx, cchgsvrx, cchgncicx, coffcodex, ctrknumx, cincx, crtcfx, crtcefx, cchgcntx, cactlitx, cdispdate, cdisptextx, cdisptypex, sntdate, fdispdate, snttextx, cnflngx, cnflngminx, cnflngmaxx, suspngx, suspminx, suspmaxx, probngx, probminx, probmaxx, courtcostx, finex, restitutionx, crtnamex, cori, corix, cori_invalid, eventdate1x)

Variable Name	Format	Length	Constraint
<u>id</u>	INT	10	PK, AUTO_INC, NOT NULL
casenum	INT	6	FK(Subject.id)
casenumx	TEXT	8	
rectypex	TEXT		
crtstatex	TEXT		
cstate	INT	2	FK(Region.id)
rapstatex	TEXT		
csource	INT	1	FK(Source.id)
ynfederalx	TEXT		
cfed	INT	1	FK(Fed.id)
cycnumx	TEXT		
cycnum	INT	3	
cseqnumx	TEXT		
cseqnum	INT	3	
crtcmtx	TEXT		
cstatnumx	TEXT		
cchglitx	TEXT		
cchgdetx	TEXT		
cchgncicx	TEXT		
coffcodex	TEXT		
cchgsvrx	TEXT		
ctrknumx	TEXT		
cincx	TEXT		
crtcfx	TEXT		
crtcefx	TEXT		
cchgcntx	TEXT		
cactlitx	TEXT		
cdispdate	TEXT		
cdisptextx	TEXT		
cdisptypex	TEXT		

Variable Name	Format	Length	Constraint
sntdatex	TEXT		
fdispdtx	TEXT		
snttextx	TEXT		
cnflgnx	TEXT		
cnflngmaxx	TEXT		
cnflngminx	TEXT		
susplngx	TEXT		
suspminx	TEXT		
suspmmaxx	TEXT		
problingx	TEXT		
probminx	TEXT		
probmaxx	TEXT		
courtcostsx	TEXT		
finex	TEXT		
restitutionx	TEXT		
crtnamex	TEXT		
cori	VARCHAR	9	
corix	TEXT		
cori_invalid	TINYINT	1	
eventdate1x	TEXT		

PRIMARY KEY id

FOREIGN KEY casenum
REFERENCES Subject (id)
FOREIGN KEY cstate
REFERENCES Region (id)
FOREIGN KEY csource
REFERENCES Source (id)
FOREIGN KEY cfed
REFERENCES Fed (id)

Notes: id will be automatically assigned on data load giving each sentence record a unique identifier.

A.4 Demographic Segment

A.4.1 Standardized Table

Demographic (id, casenum, rectype, dstate, dsource, gender, race, plcbth, dead, deadyr, deadmo, deadda, hispanic, ctzshp)

Variable Name	Format	Length	Constraint
<u>id</u>	INT	11	PK, FK(Demographicx.id), NOT NULL
casenum	INT	11	FK(Subject.id)
rectype	INT	11	FK(Segment.id)
dstate	INT	11	FK(Region.id)
dsource	INT	11	FK(Source.id)
gender	INT	11	FK(Gender.id)
race	INT	11	FK(Race.id)
plcbth	VARCHAR	2	
dead	INT	11	FK(Dead.id)
deadyr	INT	11	
deadmo	INT	11	
deadda	INT	11	
hispanic	INT	11	FK(Hispanic.id)
ctzshp	VARCHAR	2	

Key Constraints

PRIMARY KEY id

FOREIGN KEY id

REFERENCES Demographicx (id)

FOREIGN KEY casenum

REFERENCES Subject (id)

FOREIGN KEY rectype

REFERENCES Segment (id)

FOREIGN KEY dstate

REFERENCES Region (id)

FOREIGN KEY dsource

REFERENCES Source (id)

FOREIGN KEY gender

REFERENCES Gender (id)

FOREIGN KEY race

REFERENCES Race (id)

FOREIGN KEY dead

REFERENCES Dead (id)

FOREIGN KEY hispanic

REFERENCES Hispanic (id)

A.4.2 Archive Table

Demographicx (id, casenum, casenumx, rectype, demstatex, rapstatex, sexx, racex, pobx, yndeadx, dodx, hispanicx, ctzx, multistatex)

Variable Name	Format	Length	Constraint
<u>id</u>	INT	10	AUTO INC, NOT NULL
casenum	INT	6	FK(Subject.id)
casenumx	TEXT		
demstatex	TEXT		
rapstatex	TEXT		
race2x	TEXT		
pob2x	TEXT		
deadx	TEXT		
dodx	TEXT		
sex2x	TEXT		
hispanic2x	TEXT		
ctz2x	TEXT		
multistatex	TEXT		

PRIMARY KEY id

FOREIGN KEY casenum
REFERENCES Subject (demographic_id)

Notes: id will be automatically assigned on data load giving each demographic record a unique identifier.

A.5 Supervision Segment

Supervisionx (id, casenum, casenumx, rectype, supstatex, rapstatex, cycnumx, suspdatex, suptextx, sorix, admissiondtx, releasedtx, supnamex)

Variable Name	Format	Length	Constraint
<u>id</u>	CHAR	10	PK, AUTO INC, NOT NULL
casenum	INT	6	FK(Subject.id)
casenumx	TEXT		
rectype	TEXT		
supstatex	TEXT		
rapstatex	TEXT		
cycnum	INT	8	
cycnumx	TEXT		
suspdatex	TEXT		
suptext	TEXT		
sorix	TEXT		
admissiondtx	TEXT		
releasedtx	TEXT		
supnamex	TEXT		

PRIMARY KEY id

FOREIGN KEY casenum
REFERENCES Subject (id)

Notes: id will be automatically assigned on data load giving each supervision record a unique identifier.

Lookups

Admrec	(<u>id</u> , label)
Admtype	(<u>id</u> , label)
Cdv	(<u>id</u> , label)
Chg	(<u>id</u> , label)
Chg16	(<u>id</u> , label)
Chgsvr	(<u>id</u> , label)
Cnf	(<u>id</u> , label)
Dead	(<u>id</u> , label)
Dom	(<u>id</u> , label)
Dsp	(<u>id</u> , label, type)
Fed	(<u>id</u> , label)
Gender	(<u>id</u> , label)
Hispanic	(<u>id</u> , label)
Inc	(<u>id</u> , label)
Ncic	(<u>id</u> , label, type)
Oth	(<u>id</u> , label)
Pg	(<u>id</u> , label)
Phase	(<u>id</u> , label)
Pjp	(<u>id</u> , label)
Preltype	(<u>id</u> , label)
Race	(<u>id</u> , label)
RecordType	(<u>id</u> , label)
Region	(<u>id</u> , label, abbr)
Reltype	(<u>id</u> , label, type)
Segment	(<u>id</u> , label)
Shortdsp	(<u>id</u> , label)
Source	(<u>id</u> , label)
Trt	(<u>id</u> , label)
Type	(<u>id</u> , label)
Weap	(<u>id</u> , label)

Notes: Lookups were added to the database to normalize the database. The current structure of the segment tables and lookup tables enforces Third Normal Form (3NF) database normalization.

Appendix B: Recidivism Database Schema

B.1 Subject Segment

Table

Subject (id, state, moob, daob, yrob, morl, darl, yrri)

Domains

Variable Name	Format	Length	Constraint
Id	INT	11	PK, NOT NULL
State	INT	11	
Moob	INT	11	
Daob	INT	11	
Yrob	INT	11	
Morl	INT	11	
Darl	INT	11	
Yrri	INT	11	

Constraints

PRIMARY KEY id

Notes

Subject segment is the primary entity for the recidivism relational database. The id value is set to subject casenum assuming that the casenum will be able to uniquely identify each record. The total number of records should correspond to the total number of subjects in the study with one record per subject. This ensures entity integrity in Subject (no duplicate ids i.e. casenums). Since all other entities are linked to Subject id field via a foreign key constraint on the casenum field we must have at a minimum a id (casenum) for each subject inserted into the Subject table before we can begin inserting data into the other tables. This enforces referential integrity on casenum in the database (in order for a record to be added to a segment table the casenum of the record must be present in the id field within the Subject table).

B.2 Arrest Segment

Arrest (id, casenum, rectype, cycnum, cycle, rapstate, astate, aseqnum, achgseq, afed, atype, offyr, offmo, offda, arrayr, arrmo, arrda, bad_age_flag, arrname, aori, achg, achg16, ancic, acnt, ainc, achgsvr, adom, aweap, acdv, apg, adsp)

Variable Name	Format	Length	Constraint
<u>id</u>	INT	11	PK, FK(Arrestx.id), NOT NULL
casenum	INT	11	FK(Subject.id)
rectype	INT	11	FK(Segment.id)
cycnum	INT	11	
cycle	INT	11	
rapstate	VARCHAR	5	
astate	INT	11	FK(Region.id)
aseqnum	VARCHAR	3	
achgseq	INT	11	
afed	INT	11	FK(Fed.id)
atype	INT	11	FK(Type.id)
offyr	INT	11	
offmo	INT	11	
offda	INT	11	
arrayr	INT	11	
arrmo	INT	11	
arrda	INT	11	
bad_age_flag	INT	11	
arrname	VARCHAR	60	
aori	VARCHAR	9	
achg	INT	11	FK(Chg.id)
achg16	INT	11	FK(Chg16.id)
ancic	INT	11	FK(Ncic.id)
acnt	INT	11	
ainc	INT	11	FK(Inc.id)
achgsvr	INT	11	FK(Severity.id)
adom	INT	11	FK(Dom.id)
aweap	INT	11	FK(Weap.id)
acdv	INT	11	FK(Cdv.id)
apg	INT	11	FK(Pg.id)
adsp	INT	11	FK(Dsp.id)

Key Constraints

PRIMARY KEY id

FOREIGN KEY id
REFERENCES Arrestx (id)

FOREIGN KEY casenum
REFERENCES Subject (id)

FOREIGN KEY rectype
 REFERENCES Segment (id)
 FOREIGN KEY astate
 REFERENCES Region (id)
 FOREIGN KEY afed
 REFERENCES Fed (id)
 FOREIGN KEY atype
 REFERENCES Type (id)
 FOREIGN KEY ancic
 REFERENCES Ncic (id)
 FOREIGN KEY achg
 REFERENCES Chg (id)
 FOREIGN KEY achg16
 REFERENCES Chg16 (id)
 FOREIGN KEY achgsvr
 REFERENCES Severity (id)
 FOREIGN KEY ainc
 REFERENCES Inc (id)
 FOREIGN KEY adom
 REFERENCES Dom (id)
 FOREIGN KEY aweap
 REFERENCES Weap (id)
 FOREIGN KEY apg
 REFERENCES Pg (id)
 FOREIGN KEY acdv
 REFERENCES Cdv (id)
 FOREIGN KEY adsp
 REFERENCES Dsp (id)

B.3 Court-Sentence Segment

Sentence (id, casenum, rectype, cycnum, cycle, rapstatex, cstate, cseqnum, cchgseq, cfed, cdispyr, cdispmo, cdispda, crtname, cori, cchg, cchg16, cncic, ccnt, cinc, cchgsvr, cdom, cweap, ccdv, cpg, cdsp, sntyr, sntmo, sntda, sntcrt, cnf, trt, oth, incmax, incmin, suspmax, suspmin, prb, fineamt, rst, cmsrv, costs, pjp, pjp2, jpmx)

Variable Name	Format	Length	Constraint
<u>id</u>	INT	11	PK, FK(Sentencex.id), NOT NULL
casenum	INT	11	FK(Subject.id)
rectype	INT	11	FK(Segment.id)
cycnum	INT	11	
cycle	INT	11	
rapstatex	VARCHAR	5	
cstate	INT	11	FK(Region.id)
cseqnum	VARCHAR	3	
cchgseq	INT	11	
cfed	INT	11	
cdispyr	INT	11	
cdispmo	INT	11	
cdispda	INT	11	
crtname	VARCHAR	64	
cori	VARCHAR	9	
cchg	INT	11	FK(Chg.id)
cchg16	INT	11	FK(Chg16.id)
cncic	INT	11	FK(Ncic.id)
ccnt	INT	11	
cinc	INT	11	FK(Inc.id)
cchgsvr	INT	11	FK(Severity.id)
cdom	INT	11	FK(Dom.id)
cweap	INT	11	FK(Weap.id)
ccdv	INT	11	FK(Cdv.id)
cpg	INT	11	FK(Pg.id)
cdsp	INT	11	FK(Dsp.id)
sntyr	INT	11	
sntmo	INT	11	
sntda	INT	11	
sntcrt	VARCHAR	9	
cnf	INT	11	FK(Cnf.id)
trt	INT	11	FK(Trt.id)
oth	INT	11	FK(Oth.id)
incmin	DECIMAL	13,2	
incmax	DECIMAL	13,2	
suspmax	DECIMAL	13,2	
suspmin	DECIMAL	13,2	
prb	DECIMAL	13,2	
fineamt	DECIMAL	13,2	
Variable Name	Format	Length	Constraint

rst	DECIMAL	13,2	
cmsrv	DECIMAL	13,2	
costs	DECIMAL	13,2	
pjp	INT	11	FK(Pjp.id)
Pjp2	INT	11	FK(Pjp.id)
jpmx	DECIMAL	13,2	FK(Pjp.id)

Constraints

PRIMARY KEY id

FOREIGN KEY id

REFERENCES Sentencex (id)

FOREIGN KEY casenum

REFERENCES Subject (casenum)

FOREIGN KEY rectype

REFERENCES Segment (id)

FOREIGN KEY cstate

REFERENCES Region (id)

FOREIGN KEY csource

REFERENCES Source (id)

FOREIGN KEY cfed

REFERENCES Fed (id)

FOREIGN KEY cncic

REFERENCES Ncic (id)

FOREIGN KEY cchg

REFERENCES Chg (id)

FOREIGN KEY cchg16

REFERENCES Chg16 (id)

FOREIGN KEY cchgsvr

REFERENCES Severity (id)

FOREIGN KEY cinc

REFERENCES Inc (id)

FOREIGN KEY cdsp

REFERENCES Dsp (id)

FOREIGN KEY cnf

REFERENCES Cnf (id)

FOREIGN KEY trt

REFERENCES Trt (id)

FOREIGN KEY oth

REFERENCES Oth (id)

FOREIGN KEY cdom

REFERENCES Dom (id)

FOREIGN KEY cpg

REFERENCES Pg (id)

FOREIGN KEY cweap

REFERENCES Weap (id)

FOREIGN KEY ccdv

REFERENCES Cdv (id)

FOREIGN KEY pjp

REFERENCES Pjp (id)
 FOREIGN KEY pjp2
 REFERENCES Pjp2 (id)

Lookups

Admrec	(<u>id</u> , label)
Admtype	(<u>id</u> , label)
Cdv	(<u>id</u> , label)
Chg	(<u>id</u> , label)
Chg16	(<u>id</u> , label)
Chgsvr	(<u>id</u> , label)
Cnf	(<u>id</u> , label)
Dead	(<u>id</u> , label)
Dom	(<u>id</u> , label)
Dsp	(<u>id</u> , label, type)
Fed	(<u>id</u> , label)
Gender	(<u>id</u> , label)
Hispanic	(<u>id</u> , label)
Inc	(<u>id</u> , label)
Ncic	(<u>id</u> , label, type)
Oth	(<u>id</u> , label)
Pg	(<u>id</u> , label)
Phase	(<u>id</u> , label)
Pjp	(<u>id</u> , label)
Pjp2	(<u>id</u> , label)
Pretype	(<u>id</u> , label)
Race	(<u>id</u> , label)
RecordType	(<u>id</u> , label)
Region	(<u>id</u> , label, abbr)
Reltype	(<u>id</u> , label, type)
Segment	(<u>id</u> , label)
Shortdsp	(<u>id</u> , label)
Source	(<u>id</u> , label)
Trt	(<u>id</u> , label)
Type	(<u>id</u> , label)
Weap	(<u>id</u> , label)

Notes: Lookups were added to the database to normalize the database. The current structure of the segment tables and lookup tables enforces Third Normal Form (3NF) database normalization.