Senzing
Generic Entity Specification

Version: *2.8.2*

Date: *August 1, 2022*

Table of Contents

Overview

The Senzing engine performs Entity Resolution to determine when entities are the same or related within and across data sources.

This specification focuses on entities that are persons or organizations, such as customers, prospects, vendors, employees, and watch lists. It contains a dictionary of pre-configured attributes that are used to resolve and relate persons or companies and outlines the process of creating data sets with them so that they are readily consumable by the Senzing engine. The dictionary also serves to identify what information is desirable to perform Entity Resolution.

Data must be presented to the engine in either a JSON or CSV file format using the dictionary of registered attributes contained in this specification. The advantage of JSON is that its hierarchical structure allows for multiple names, addresses, phones, etc to be presented in a single structure as one record may have only one address and another may have five. CSV files are flat and multiple values must be presented as additional columns, so if the maximum number of addresses is five, then all rows in the csv file will include space for five addresses, whether needed for that record, or not.

There are several ways to load data into Senzing …

- You can install the product directly and use the G2Loader.py python script to load files of json records as demonstrated here … Exploratory-Data-Analysis-1-Loading-the-truth-set-demo
- You can set up a docker environment that loads from queues as demonstrated here… https://github.com/Senzing/docker-compose-demo
- Or you can call Senzing directly from your current processes to add, update, delete or even search records on demand by calling the api directly as documented here … docs.senzing.com

In any of these cases, your source records must be mapped according to this specification.

Mapping source data

Types of data sources

Senzing entity resolution finds matches within and across data sources. There are two types of data sources you may want to resolve:

1. A master list of entities such as a customer list, employee list, etc. While this is usually a curated list of unique entities, duplicates may be expected and are often found. As well, you may be loading them in order to see if they have any connections to other data sources such as watchlists, or to supplement or verify them with reference data from external data providers.

2. An event or transaction that contains the identifying information about the entity rather than a link to a master record as defined above. For instance, if your registered customer sends or receives money to an unknown external party, those external parties can be resolved into unique entities so their activities can be accumulated and connections can be made to known entities and watch lists.

   When processing events or transactions with external parties, it is best to use a hash of the identifying information for a party as its key and only send parties to Senzing for resolution. The event or transaction itself should be stored outside of Senzing with a linking table that connects them to external party keys sent to Senzing for resolution. The transactions can then be joined to Senzing's resolved entity IDs for aggregation and decision making.

Updating vs replacing records

When information about a master record changes, that record should be re-sent to Senzing for resolution. But was the change an update or a correction? Was the prior address wrong, or did the entity move. Are you even allowed to keep the prior data - likely not on a watch list! These are issues that are normally addressed by the source systems themselves. For this reason, when given the same key (known as a record_id in Senzing), the record is replaced rather than updated.

Therefore, all the attributes of an entity must be presented in a single json document including any historical values kept by the source system. If an entity is comprised of data in several different tables, those tables must be joined together so the entire entity can be presented at once. We use a json format so that lists of child records such as additional or prior names, addresses, and identifiers can be presented as json lists within the single document for the entity.

Desired attributes for a person

The sample_person.csv attached to this article can be used to map your data to if desired. There is also a corresponding sample_person.json file if you prefer. These files contain the likely fields you will run into when mapping persons to the generic entity format. In fact, you should try to map as many of these fields as possible. The sample person structure contains fields for …

- Primary name
- Date of birth and gender
- Passport, driver's license, social security number, national insurance number
- Home and mailing addresses
- Home and cell phone numbers
- Email and social media handles
- Groups that they are associated with such as their employer name
- Any key dates, statuses or amounts that can help you find meaning in the matches. For instance, a vendor related to an employee who has influence over purchases is more important than the same vendor related to an employee that doesn't.

Desired attributes for an organization

The sample_organization.csv attached to this article can be used to map your data to if desired. There is also a corresponding sample_organization.json file if you prefer. These files contain the likely fields you will run into when mapping organizations to the generic entity format. In fact, you should try to map as many of these fields as possible. The sample organization structure contains fields for …

- Primary name
- National Registry numbers
- Tax ID numbers
- Other ID numbers assigned by agencies or data providers
- Primary and mailing addresses
- Primary and other phone numbers
- Website and social media handles
- Any key dates, statuses or amounts that can help you find meaning in the matches. For instance, a current company you do business with who is on a watch list for bad reasons is more important than the same match to a company you did business with several years ago.

Creating JSON files

Below is the basic structure of a json record the engine can consume. Note that most attributes are at the root level. However, lists must be used when there are multiple values for the same attributes.

```
{
	"DATA_SOURCE": "CUSTOMER ",                 ← a data source code is required
	"RECORD_ID": "1001",                        ← a record ID is always desired
	"RECORD_TYPE": "PERSON",                    ← a record type is also desired
	"NAME_LIST": [{                             ← multiple name segments placed in a list
		"NAME_TYPE": "PRIMARY",
		"NAME_LAST": "Adams",
		"NAME_FIRST": "Allen",
		"NAME_MIDDLE": "A"
	},
	{
		"NAME_TYPE": "AKA",
		"NAME_LAST": "Adams",
		"NAME_FIRST": "Archie"
	},
	"DATE_OF_BIRTH": "2/1/1981",                ← single values do not need to be placed in a list
	"SSN_NUMBER": "111-11-1111",
	"ADDRESS_LIST": [{                          ← multiple address segments placed in a list
		"ADDR_TYPE": "HOME",
		"ADDR_LINE1": "111 First St",
		"ADDR_CITY": "Las Vegas",
		"ADDR_STATE": "NV",
		"ADDR_POSTAL_CODE": "89111"
	},
	{
		"ADDR_TYPE": "MAIL",
		"ADDR_LINE1": "PO Box 111",
		"ADDR_CITY": "Las Vegas",
		"ADDR_STATE": "NV",
		"ADDR_POSTAL_CODE": "89111"
	}],
	"CUST_SINCE_DATE": "2/1/2012",              ← values not used for resolution, only at the root level
	"CUST_STATUS": "Active",
	"AGE_BRACKET": "30-40",
	"INCOME_LEVEL": "100+"
}
```

When creating a file of JSON messages for the G2Loader program, each message should be on one line. It is expecting a file of JSON messages, like so …

```
{"DATA_SOURCE": "CUSTOMER ", "RECORD_ID": "1001", "NAME": [{"NAME_TYPE": "PRIMARY", "NAME_LAST": "Adams", …
{"DATA_SOURCE": "CUSTOMER ", "RECORD_ID": "1002", "NAME": [{"NAME_TYPE": "PRIMARY", "NAME_LAST": "Baker", …
{"DATA_SOURCE": "CUSTOMER ", "RECORD_ID": "1003", "NAME": [{"NAME_TYPE": "PRIMARY", "NAME_LAST": "Cooper", …
```

Creating CSV files

Mapping csv files is normally accomplished by replacing the column header names with the registered attributes names contained in this specification. Column names in a CSV must be unique, so multiple values for attributes such as multiple names, multiple addresses, etc must be prefixed with a term denoting the type of name, address, etc. Here are the list of column headers that could be used to flatten out the JSON example above …

DATA_SOURCE	← defaults to the setting in the project file if not supplied on every record
RECORD_ID	← a record ID is always desired
RECORD_TYPE	← a record type is also desired
PRIMARY_NAME_LAST	← name type value moved to column prefix
PRIMARY_NAME_FIRST	
PRIMARY_NAME_MIDDLE	
AKA_NAME_LAST	← name type value moved to column prefix
AKA_NAME_FIRST	
AKA_NAME_MIDDLE	
DATE_OF_BIRTH	
SSN_NUMBER	
HOME_ADDR_LINE1	← address type value moved to column prefix
HOME_ADDR_CITY	
HOME_ADDR_STATE	
HOME_ADDR_POSTAL_CODE	
MAIL_ADDR_LINE1	← address type value moved to column prefix
MAIL_ADDR_CITY	
MAIL_ADDR_STATE	
MAIL_ADDR_POSTAL_CODE	
CUST_SINCE_DATE	← fields not used for resolution can still be included with their original names
CUST_STATUS	
AGE_BRACKET	
INCOME_LEVEL	

A mapped csv file actually looks this and is usually accomplished by simply replacing the current column headers with corresponding attribute labels and names in this specification …

```
RECORD_ID, PRIMARY_LAST_NAME, PRIMARY_FIRST_NAME, PRIMARY_MIDDLE_NAME …
1001,Adams,Alan,  …
1002,Baker,Betty, …
1003,Cooper,Cindy, …
```

Dictionary of registered attributes

Attributes for the record key

Senzing is an entity repository that helps locate records for the same entity across data sources. Think of it as a pointer system to where an entity's records can be found. These are the fields required to tie the records in Senzing back to the contributing sources.

Attribute name	Type	Required	Example	Notes
DATA_SOURCE	String	Required	CUSTOMER	This is an important designation for reporting. For instance, you may want to know how many customers are on watch lists, or how many customers in one data source match customers from another. Choose your data source codes based on how you want your reports to appear.
RECORD_ID	String	Strongly Desired	1001	This value must be unique within a data source and is used to add new or replace records with updated values. Because the smallest unit of update is a record, all of the attributes for a record_must be presented together including any historical addresses, phone numbers, etc you want to keep on the record.
RECORD_TYPE	String	Desired	PERSON/ ORGANIZATION	This attribute helps prevent two different types of records from resolving to each other while still allowing relationships between them. Be sure to use standardized terms like PERSON and ORGANIZATION across all your data sources.

Important notes:

- The DATA_SOURCE default length limit is 25 characters. Please email support@senzing.com if you need to increase the length limit.
- Caution: If you do not supply a record_id, one will be generated based on a hash of the identifying attributes effectively rendering updates impossible. If you do not supply a unique record_id, you should not load the same set of records more than once.

Attributes for names of individuals or organizations

A name is a highly desirable feature to map. Most resolution rules will require a matching name.

Attribute name	Type	Example	Notes
NAME_TYPE	String	PRIMARY, ALIAS	Most data sources have only one name, but when there are multiple, there is usually one primary name and the rest are aliases. Whatever terms are used here should be standardized across the data sources included in your project.
NAME_FULL	String	Robert J Smith	This is the full name of an individual. It should only be populated when the parsed name of an individual is not available, although parsed names for an individual are most desirable. The system will not allow both a full name and the parsed names to be populated in the same set of name fields. [See handling duplicate columns later in this document.]
NAME_ORG	String	Acme Tire Inc.	This is the organization name.
NAME_LAST	String	Smith	This is the last or sur name of an individual.
NAME_FIRST	String	Robert	This is the first or given name of an individual.
NAME_MIDDLE	String	J	This is the middle name of an individual.
NAME_PREFIX	String	Mr	This is a prefix for an individual's name such as the titles: Mr, Mrs, Ms, Dr, etc.
NAME_SUFFIX	String	MD	This is a suffix for an individual's name and may include generational references such as: JR, SR, I, II, III and/or professional designations such as: MD, PHD, PMP, etc.

Important notes:

- The "PRIMARY" **NAME_TYPE** helps select the best name to display for an entity. See Special attribute types and labels for when to use this. It is best to always specify name type!
- The **NAME_FULL** attribute is provided if the parsed name fields are unavailable. You would not map both a **NAME_FULL** and any other name fields in the same name segment.
- If there is a common or nick name field, it represents a "second" name the individual is known by. In this case, map a second set of name columns duplicating the last name with the common name.
- If using **NAME_ORG** then this record should be about an organization, not an individual *i.e.,* do not map any of the individual name fields. You would not map both a **NAME_ORG** and any other name fields in the same name segment.
- Sometimes there is both an organization name and a person name on a record, such as a contact list where you have the person and who they work for. In this case, you would map the person's name as a name and the company as their employer. See Attributes for group associations for more information on this important distinction.

Attributes for addresses

Addresses are important, especially when identifiers are not available. One of the more common resolutions will be made on name and address.

Attribute name	Type	Example	Notes
ADDR_TYPE	String	HOME	This is a code that describes how the address is being used such as: HOME, MAILING, BUSINESS*, etc. Whatever terms are used here should be standardized across the data sources included in your project.
ADDR_FULL	String		This is a single string containing the all address lines plus city, state, zip and country. Sometimes data sources have this rather than parsed address. Only populate this field if the parsed address lines are not available.
ADDR_LINE1	String	111 First St	This is the first address line and is required if an address is presented.
ADDR_LINE2	String	Suite 101	This is the second address line if needed.
ADDR_LINE3	String		This is the third address line if needed.
ADDR_LINE4	String		This is the fourth address line if needed.
ADDR_LINE5	String		This is the fifth address line if needed.
ADDR_LINE6	String		This is the sixth address line if needed.
ADDR_CITY	String	Las Vegas	This is the city of the address.
ADDR_STATE	String	NV	This is the state or province of the address.
ADDR_POSTAL_CODE	String	89111	This is the zip or postal code of the address.
ADDR_COUNTRY	String	US	This is the country of the address.
ADDR_FROM_DATE	Date	2016-01-14	This is the date the entity started using the address if known. It is the used to determine the latest value of this type being used by the entity.
ADDR_THRU_DATE	Date		This is the date the entity stopped using the address if known.

Important notes:

- The **ADDR_FULL** attribute is provided if the parsed address fields are unavailable. You would not map both an **ADDR_FULL** and any other address fields in the same address segment.
- The "BUSINESS" **ADDR_TYPE** adds weight to physical business addresses. See Special attribute types and labels for when to use this.

Attributes for phone numbers

Like addresses, phone numbers can be important, especially when identifiers are not available. A common resolution will be based on name, phone, and date of birth.

Attribute name	Type	Example	Notes
PHONE_TYPE	String	MOBILE	This is a code that describes how the phone is being used such as: HOME, FAX, MOBILE*, etc. Whatever terms are used here should be standardized across the data sources included in your project.
PHONE_NUMBER	String	111-11-1111	This is the actual phone number.
PHONE_FROM_DATE	Date	2016-01-14	This is the date the entity started using the phone number if known. It is the used to determine the latest value of this type being used by the entity.
PHONE_THRU_DATE	Date		This is the date the entity stopped using the phone number if known.

Important notes:

- The "MOBILE" phone type adds weight to mobile phones. See Special attribute types and labels for when to use this.

Attributes for physical and other attributes

Physical attributes can like DATE_OF_BIRTH help reduce over matching (false positives). Usually gender and date of birth are available and should be mapped if possible.

Attribute name	Type	Example	Notes
GENDER	String	M	This is the gender such as M for Male and F for Female.
DATE_OF_BIRTH	String	1980-05-14	This is the date of birth for a person and partial dates such as just month and day or just month and year are ok.
DATE_OF_DEATH	String	2010-05-14	This is the date of death for a person. Again, partial dates are ok.
NATIONALITY	String	US	This is where the person was born and should contain a country name or code
CITIZENSHIP	String	US	This is the country the person is a citizen of and should contain a country name or code.
PLACE_OF_BIRTH	String	US	This is where the person was born. Ideally it is a country name or code. However, they often contain city names as well.
REGISTRATION_DATE	String	2010-05-14	This is the date the organization was registered, like date of birth is to a person.
REGISTRATION_COUNTRY	String	US	This is the country the organization was registered in, like place of birth is to a person.

Attributes for government issued identifiers

Government issued IDs help to confirm or deny matches. The following identifiers should be mapped if available.

Attribute name	Type	Example	Notes
PASSPORT_NUMBER	String	123456789	This is the passport number.
PASSPORT_COUNTRY	String	US	This is the country that issued the ID.
DRIVERS_LICENSE_NUMBER	String	123456789	This the driver's license number.
DRIVERS_LICENSE_STATE	String	NV	This is the state or province that issued the driver's license.
SSN_NUMBER	String	123-12-1234	This is the US Social Security number.
SSN_LAST4	String	1234	This is just the last4 digits of the SSN for use when the full SSN is not available.
NATIONAL_ID_NUMBER	String	123121234	This is the national insurance number issued by many countries. It is similar to an SSN in the US.
NATIONAL_ID_COUNTRY	String	CA	This is the country that issued the ID.
TAX_ID_TYPE	String	EIN	This is the tax id number for a company, as opposed to an SSN or NIN for an individual.
TAX_ID_NUMBER	String	123121234	This is the actual ID number.
TAX_ID_COUNTRY	String	US	This is the country that issued the ID.
**** OTHER_ID_TYPE***	String	CEDULA	This is the type of any other identifier, such as registration numbers issued by other authorities than listed above.
**** OTHER_ID_NUMBER***	String	123121234	This is the actual ID number.
OTHER_ID_COUNTRY	String	MX	This is the country that issued the ID number.
TRUSTED_ID_TYPE	*String*	*TRUE_SSN*	*The type of ID that is to be trusted. See the note below*
TRUSTED_ID_NUMBER	*String*	*123-45-1234*	*The trusted unique ID.*

Important notes:

- A **TRUSTED_ID** is a very special identifier that will resolve records together even if they have different names, dobs, or other identifiers. For example, if the SSN of a data source is so trusted it should resolve records despite other differences, it can also be mapped as a **TRUSTED_ID_NUMBER** with the **TRUSTED_ID_TYPE** of "SSN" to resolve within and across data sources that are so trusted.
- A **TRUSTED_ID** can also be used to manually force records together or apart as described here… https://senzing.zendesk.com/hc/en-us/articles/360023523354-How-to-force-records-together-or-apart
- *Use * **OTHER_ID** sparingly!   It is just a catch all for identifiers you know nothing about but still want to use to help match.   Therefore if you know anything about an identifier not listed above, you should add it as its own identifier as described here …* How to add a new identifier.

Attributes for identifiers issued by organizations

The following identifiers have been added over time and can also be mapped if available.

Attribute name	Type	Example	Notes
ACCOUNT_NUMBER	String	1234-1234-1234-1234	This is an account number such as a bank account, credit card number, etc.
ACCOUNT_DOMAIN	String	VISA	This is the domain the account number is valid in.
DUNS_NUMBER	String	123123	The unique identifier for a company https://www.dnb.com/duns-number.html
NPI_NUMBER	String	123123	A unique ID for covered health care providers. https://www.cms.gov/Regulations-and-Guidance/Administrative-Simplification/NationalProvIdentStand/
LEI_NUMBER	String	123123	A unique ID for entities involved in financial transactions. https://en.wikipedia.org/wiki/Legal_Entity_Identifier

Attributes for websites, emails and other social handles

The following social media attributes are available.

Attribute name	Type	Example	Notes
WEBSITE_ADDRESS	String	somecompany.com	This is a website address, usually only present for organization entities.
EMAIL_ADDRESS	String	someone@somewhere.com	This is the actual email address.
LINKEDIN	String	xxxxx	This is the unique identifier in this domain.
FACEBOOK	String	xxxxx	This is the unique identifier in this domain.
TWITTER	String	xxxxx	This is the unique identifier in this domain.
SKYPE	String	xxxxx	This is the unique identifier in this domain.
ZOOMROOM	String	xxxxx	This is the unique identifier in this domain.
INSTAGRAM	String	xxxxx	This is the unique identifier in this domain.
WHATSAPP	String	xxxxx	This is the unique identifier in this domain.
SIGNAL	String	xxxxx	This is the unique identifier in this domain.
TELEGRAM	String	xxxxx	This is the unique identifier in this domain.
TANGO	String	xxxxx	This is the unique identifier in this domain.
VIBER	String	xxxxx	This is the unique identifier in this domain.
WECHAT	String	xxxxx	This is the unique identifier in this domain.

Attributes for group associations

Groups a person belongs to can also be useful for resolving entities. Consider two contact lists that only have name and who they work for as useful attributes.

Attribute name	Type	Example	Notes
EMPLOYER_NAME	String	ABC Company	This is the name of the organization the person is employed by.
GROUP_ASSOCIATION_TYPE	String	MEMBER	This is the type of group an entity belongs to.
GROUP_ASSOCIATION_ORG_NAME	String	Group name	This is the name of the organization an entity belongs to.
GROUP_ASSN_ID_TYPE	String	DUNS	When the group a person is associated with has a registered identifier, place the type of identifier here.
GROUP_ASSN_ID_NUMBER	String	12345	When the group a person is associated with has a registered identifier, place the identifier here.

Important Notes:

- Group associations should not be confused with disclosed relationships described later in this document.   Group associations help resolve entities whereas disclosed relationships help relate them.
- If all you have in common between two data sources are name and who they work for, a group association can help resolve the Joe Smiths that work at ABC company together.
- Group associations are subject to generic thresholds to help reduce false positives and keep the system fast.   Therefore they will not help resolve *all* the employees of a large company across data sources. But they could help to resolve the smaller groups of executives, primary contacts, or owners of large companies across data sources.

Attributes for disclosed relationships

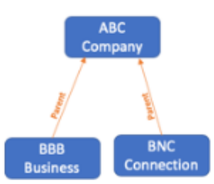
Some data sources keep track of known relationships between entities, such as familial relationships and company hierarchies.   This structure allows you to tell G2 about such relationships. Look for a table within the source system that defines such relationships and include them here.

For instance, if the relationship says that customer 1001 is the SPOUSE of customer 1002, then customer 1001 should be given an anchor domain and key of 1001 and customer 1002 should be given a rel_pointer_domain and key that "points" to back to customer 1001 as its spouse.   This is all it takes to create a disclosed relationship.

However, some source systems point each record to the other with a more descriptive role.   For instance, if customer 1001 is pointed to customer 1002 as the father and customer 1002 is pointed to customer 1001 as the son then both customers should have a rel_anchor_domain and key for itself as well as a rel_pointer_domain, key and role that relates it to the other.

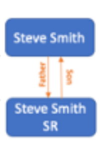
Attribute name	Type	Example	Notes
REL_ANCHOR_DOMAIN	String	CUSTOMER_ID	This code describes the domain of the rel_anchor_key. The key must be unique within the domain. For instance, customer systems might use the customer_id to define relationships.
REL_ANCHOR_KEY	String	1001	The rel_anchor_key along with the associated domain comprises a unique value that other records can "point" to in order to create a disclosed relationship.
REL_POINTER_DOMAIN	String	CUSTOMER_ID	See rel_anchor_domain above.
REL_POINTER_KEY	String	1001	See rel_anchor_key above. A rel_pointer_domain and key on one record point to a rel_anchor_domain and key on another record to in order to create a relationship between them.
REL_POINTER_ROLE	String	SPOUSE	This is the role the anchor record plays in relationship to the pointer record. *Note: Be careful not to use very long names here as so they are should appear on the line between two nodes on a graph.*

Example company hierarchy …



DATA_SOURCE	RECORD_ID	NAME_ORG	REL_ANCHOR_DOMAIN	REL_ANCHOR_KEY	REL_POINTER_DOMAIN	REL_POINTER_KEY	REL_POINTER_ROLE
COMPANIES	1001	ABC Company	COMPANY_ID	1001			
COMPANIES	1002	BBB Business	COMPANY_ID	1002	COMPANY_ID	1001	PARENT
COMPANIES	1003	BNC Connections	COMPANY_ID	1003	COMPANY_ID	1001	PARENT

Example bi-directional familial relationship …



DATA_SOURCE	RECORD_ID	NAME_FULL	REL_ANCHOR_DOMAIN	REL_ANCHOR_KEY	REL_POINTER_DOMAIN	REL_POINTER_KEY	REL_POINTER_ROLE
CUSTOMER	1001	Steve Smith	CUSTOMER_ID	1001	CUSTOMER_ID	1002	FATHER
CUSTOMER	1002	Steve Smith Sr	CUSTOMER_ID	1002	CUSTOMER_ID	1001	SON

Attributes for values that are not used for entity resolution

Sometimes it is desirable to include additional attributes that can help determine the importance of a resolution or relationship. These attributes are not used for entity resolution because they are not configured in Senzing. These attributes may include values such as additional dates, statuses, types, flags, or aggregated amounts at the entity level.

For example:

- The LIFETIME_VALUE of a customer can help determine what kind of discount should be applied to their order or if a new customer is related to a high value customer.
- The TERMINATION_REASON of an employee can help determine if a new job applicant should be hired or not.
- The BUSINESS_RISK or GEOGRAPHICAL_RISK of a customer may help determine if a high dollar transaction should be reviewed before it is executed.
- A vendor related to an employee who has influence over purchases is more important than the same vendor related to an employee that doesn't.
- A current company you do business with who is on a watch list for bad reasons is more important than the same match to a company you did business with several years ago.

On smaller Senzing systems, you may want to include and store additional non-Senzing attributes. On larger Senzing systems, it is best practice to load only the configured attributes used for entity resolution, and use a data warehouse or other external system to access additional non-Senzing attributes.

Special attribute types and labels

Some features have special labels that add weight to them. For instance, you might find a whole family at a "home" address, but only one company (or company facility) at its physical "business" address. The following special labels can be used to augment a feature's weight …

Feature	Label	Notes	When to use
NAME	PRIMARY	People can have aka's and nick names; companies can have dbas. When the system resolves multiple records into an entity, the most complete "primary" name will be chosen over any other type.	*Usage: often* To help select the best name to display for an entity
ADDRESS	BUSINESS	Companies with multiple facilities or outlets often share corporate phone numbers and website addresses. Use this label to help break matches based on their physical location.	*Usage: often* To prevent overmatching of companies.
PHONE	MOBILE	Home and work phone numbers are usually shared. Use this label to add weight to mobile or "cell" phones as they are shared far less often.	*Usage: rare* Only apply if data source reliably uses mobile phones to distinguish entities.

How to use:

Labels are either used as an attribute prefix such as:

```
…
"BUSINESS_ADDR_LINE1": "111 First St ",
"BUSINESS_ADDR_CITY": "Anytown",
…
```

Or by its "type"attribute in a json list such as:

```
"ADDRESS_LIST": [{
        "ADDR_TYPE": "BUSINESS", ←
        "ADDR_LINE1": "111 First St",
        "ADDR_CITY": "Anytown",
        …
```

Project file for G2Loader

A project file is not necessary, but the G2Loader.py python script can either load files one by one with a -f parameter or multiple files listed in a project file with the -p parameter. The project file itself may either be a JSON file or a CSV file using the following column or tag names.

Attribute name	Type	Example	Notes
DATA_SOURCE	String	CUSTOMER	This is an important designation for reporting. For instance, you may want to know how many customers are on watch lists, or how many customers in one data set match customers from another. Choose your data source codes based on how you want your reports to appear.
FILE_FORMAT	String	CSV	This is the format of the files for this source. Valid values are JSON, CSV, TAB, or PIPE.
FILE_NAME	String	custdump_20160511.csv	This is the name of the file that contains the data for this data source.

Additional configuration

Senzing comes pre-configured with all the features, attributes, and settings you will likely need to begin resolving persons and organizations immediately. The only configuration that really needs to be added is what you named your data sources.

The way you configure Senzing is through the G2ConfigTool.py script located on the python folder in the directory tree for your project. To use it, go to the python folder and type …

G2ConfigTool.py

Then type "help" at the prompt. There is a lot you can do in there, but most of it you should not use unless directed to do so by Senzing support. For instance, adding new rules or adjusting thresholds should not be attempted without first contacting support for guidance.

However, you will often use this tool to add new data sources and sometimes to add new identifiers that are not in our default for configuration.

How to add a data source

Adding a new data source is a simple as registering the code you want to use for it. Most of the reporting you will want to do is based on matches within or across data sources. For instance …

- If you want to know when a customer record matches a watchlist record, you should have a data source named "CUSTOMER" and another one named "WATCHLIST".
- Or if you are matching two customer data sources to find the overlap, you should have one data source named CUSTOMER1 and another named CUSTOMER2 or to be more descriptive you might name them based on the line of business such as "BANKING-CUSTOMER" and "MORTGAGE-CUSTOMER".

To add a new data source named "CUSTOMER", at the the G2ConfogTool prompt type …

addDataSource CUSTOMER

How to add a new identifier

Occasionally you will need to add a new kind of identifier. This consists of adding a feature and its associated attributes and there are templates to help you. At the G2ConfigTool prompt type "templateAdd list" to see the current list of templates.

```
root@480dc4e4e418:/opt/senzing/g2/python# ./G2ConfigTool.py

Initializing Senzing engines...

Welcome to G2Config Tool. Type help or ? to list commands.

(g2cfg) templateAdd list

        GLOBAL_ID - globally unique identifer (like an ssn, a credit card, or a medicare_id)
               behaviors: ['F1', 'F1E', 'F1ES']
               comparisons: ['ID_COMP', 'EXACT_COMP']

        STATE_ID - state issued identifier (like a drivers license)
               behaviors: ['F1', 'F1E', 'F1ES']
               comparisons: ['ID_COMP']

        COUNTRY_ID - country issued identifier (like a passport)
               behaviors: ['F1', 'F1E', 'F1ES']
               comparisons: ['ID_COMP']

(g2cfg)
```

The Senzing engine performs behavior based matched which you can read more about here … [Principle-Based-Entity-Resolution]

For identifiers, the three behaviors you can choose from are …

- **F1** is the default and means that only one person should have this identifier, but it is possible or even likely for them to have two. A match on name and address will not be broken there are conflicting F1s
- **F1E** adds the "exclusive" behavior which means they should only have one. A match on name and address will be broken if each record has a different F1E
- **F1ES** – Adds the "exclusive" and "stable" behaviors which adds more weight to the identifier. It will not only break matches, but it will help cement them. If two records share an F1ES but have a conflicting F1E, the match will not be broken. Conversely, if two records share an F1E but have a conflicting F1ES the match will be broken.

For identifiers, the two comparisons you can choose from are …

- **ID_COMP** is the default and does a fuzzy comparison of identifier strings.
- **EXACT_COMP** which requires the identifier strings to match exactly. It is rare that you will want to switch to this as transposition errors in human entered strings are quite common.

While that is a lot to take in, look at it like this …

- If you want the new identifier to break matches, make it an **F1E**
- If you want the new identifier to help ensure a match, make it an **F1ES**
- And if you are not entirely sure that it should do either, make it a plain **F1**. It is pretty rare to find reliable **F1E** and **F1ES** identifiers other than government issued ones we come pre-configured with.

Most new identifiers will be added like this …

templateAdd {"feature": "MY_NEW_ID", "template": "global_id"}

This will create a feature and an attribute you can map to called: **MY_NEW_ID**. It will help make matches; but will not help break matches. A person or company can have more than one of them.

Since ID_COMP is the default comparison, transposition errors will be considered close. If you want to force an exact comparison, you would add it like this …

templateAdd {"feature": "MY_NEW_ID", "template": "global_id", “comparison”: “EXACT_COMP}

If you have a state bounded identifier, where the same ID number might be issued by different states you would add it like this …

templateAdd {"feature": "MY_NEW_ID", "template": "state_id"}

This will create a feature called **MY_NEW_ID** with the attributes: **MY_NEW_ID_NUMBER** and **MY_NEW_ID_STATE** that you can map to.

It can only compared with ID_COMP due to the fact that some sources might supply the state field while others may not and human entered ID numbers and state codes are often mistyped.

If you wanted to add weight to this new identifier, you would add it like this …

templateAdd {"feature": "MY_NEW_ID", "template": "state_id", “behavior”: “F1E}

And for a country bounded identifier, you would just replace the template **state_id** with **country_id** above.