Chapter 8
8.0 Overview of Alam-Padenga Process Model

Definition

Alam-Padenga process model is a dynamic and intelligent linking of an application software's variable names. The variable names to be linked can reside in disparate application modules. By using the Alam-Padenga process model and intelligently linking this to source code, an application software package can provide a more complete and intelligent view of this information. Furthermore, these linked variable names can be used to provide a 360-degree view of an application package, to support variable name interactions, and to uncover trends and patterns that provide insight into achieving reduced or optimum reengineering costs.

Reengineering applications where Alam-Padenga process model can be critical, profitable and can influence both profit and satisfaction is almost limitless. Some of these areas include:

- Legacy Systems
- Health care application software
- Financial and insurance application software
- Pharmaceutical application software
- Hospitality applications software
- Public applications software
- Transportation and logistics application software
- Manufacturing applications software
- Automotive application software
8.1 ALAM-PADENGA PROCESS MODEL Background Scenario

The following hypothetical scenario helps illustrate the role of Alam-Padenga process model. Application software systems generally implemented in a phased manner, e.g. one department at a time. Over a period of ten years, a variable named “localSingleUnderline” was used in various modules of an overall application package. Each module maintained a separate database with relevant details of variable name “localSingleUnderline” and other similar variable names. During the course of the application package implementation and maintenance, “localSingleUnderline” was consequently changed several times. Each time a maintenance exercise was conducted, “localSingleUnderline” was given new definitions and characteristics related to the current module in question. Thus some modules stored the old definitions and characteristics whilst others recorded newer definitions and characteristics. Over the years several other variable names with name combinations of “local”, “localSingle”, “SingleUnderline”, “underline”, also had their definitions and characteristics changed during implementation and maintenance across the modules in the application package. Some of these variable names also had their pointer address characteristics changed.

After several years of maintenance improvements, a team of developers require the complete variable names history of “localSingleUnderline” from across all modules in the application package to evaluate the potential problem in single line underlining, in order to solve an underlining conflict that has arisen. Typically when a variable name is declared in a module, personally identifiable information is built around it, as reference for future maintenance. Most modules take only the relevant variable name information and store it within the source code file as comments. For instance, each module developed by different developers and maintained over several years by different developers and consultants may end up
storing a variable name "name" in a different format. The Finance module had stored "localSingleUnderline" as "LocalsingleUnderline" while the Administration module stored this name as "local_Single_Underline". Thus, after a few years, when you want a complete variable name history of "localSingleUnderline", from across all modules, the first challenge would be to identify the correct variable name record of variable name "localSingleUnderline" from among all aliases variable name records in each module.

For example, how would you differentiate between two variable name records in the maintenance module, which have the same pointer address characteristics as "local_Single_Underline"? How can you be sure a particular variable name record belongs to this "localSingleUnderline" and not to another variable name called "LocalsingleUnderline"? The solution is to compare "localSingleUnderline" other identifiable attributes (such as pointer address, scope, inheritance, et cetera) with those of "localSingleUnderline".

8.2 Background Information

Variable name information is fragmented. Each module had partial information of "localSingleUnderline". Besides this name, some modules may have stored only its pointer address and scope, and while others stored its inheritance, polymorphism, et cetera. Due to this situation, none of the modules had a complete set of details regarding "localSingleUnderline". Duplicate records exist. Duplicate records were created when "localSingleUnderline" was altered during consultancy or maintenance by different developers over several years.
As mentioned above, these individual modules typically have their own variable naming comments embedded in their source code, which present the following challenges:

- Source code comments are not linked with each other
- The source code across these modules may be different (through maintenance)
- Source code structures are also different

To compound the problem, there are several other variable names sharing the name “local” or “Single” or “Underline” within the multiple source code modules. Some of these members have details and attributes similar to those of “localSingleUnderline”. Any proposed process modelling solution should not only break the barriers between the disparate systems of the modules, but also compare “localSingleUnderline”’s personally identifiable attributes with the attributes of other members and accurately identify “localSingleUnderline”’s variable name records in each of the multiple modules. It should then link these respective records in the multiple modules and source code comments so that a complete profile of “localSingleUnderline” is available to the developers, and to all modules.

Note: In Alam-Padenga process model, we use only the personally identifiable attributes of members to identify and link variable name member records.

8.3 Alam-Padenga Identity Hub Process Model In Action

Alam-Padenga Identity Hub process model is deployed to resolve this situation.

- Alam-Padenga Identity Hub receives variable names data in real-time or batch from individual systems via relational database connections, APIs, web services, messaging solutions, or flat file formats. These "source"
systems provide the data used for scoring, matching, and linking variable name members within Alam-Padenga Identity Hub solution.

- After data from the source systems is received by Alam-Padenga Identity Hub module, it is run through a derivation routine. Derivation is the process of extracting demographic variable name data elements to be used in scoring and matching. The derived variable name data is then stored in a highly-optimized format in a relational database.

- If there are any exceptions, or if variable name data entities fall outside the defined matching parameters, then a flag will be raised to mark these as possible matches and create tasks for later review and resolution. This process is also known as data remediation and is fully supported.

This Alam-Padenga process model process enables establishment of “localSingleUnderline”’s identity and accurately links its variable name history records from across the various sources (databases).

8.4 Benefits

By implementing the Alam-Padenga Identity Hub process model, the following benefits are yielded:

- Alam-Padenga Identity Hub accurately differentiates “localSingleUnderline”’s variable name history records from records of numerous other variable names with similar details.

- Developers and consultants get a clear picture of the variable name’s variable name history.

- A 360-degree view of each variable name is now available on demand to all modules in the application package.

- Clear insights on trends are now available on demand.
• Next time a change to "localSingleUnderline" is made to its pointer address or other information at one module, all other modules can receive the update.

The following graphic illustrates the concept of a 360-degree view of member data.

8.5 Alam-Padenga Identity Hub Overview

Alam-Padenga Identity Hub accurately identifies and links the variable name records of all modules so as to drive the on-demand enterprise. Alam-Padenga Identity Hub can help any volume of variable name data into the clean, complete variable name profiles in an effort to drive better, more cost-effective and robust interactions with application packages. The concepts of scoring, matching, and linking are discussed in greater detail in the following sections. In summary, Alam-Padenga Identity Hub:

• Breaks down variable naming barriers between application packages source code
• Links variable name data on-demand, be it in real-time, near real-time, or batch
• Delivers the most accurate results
• Works with any volume of variable name data
• Works with variable name data from any source
• Works with any type of structured variable name data
• Does not force standardization of variable name data or require changes to variable name application identifiers in any database or system
• Matches implementation to local needs
8.6 ARCHITECTURAL AND TECHNICAL OVERVIEW

8.6.1 Components of Alam-Padenga Identity Hub

Alam-Padenga Identity Hub is the critical link between variable name data sources and an operational application. Alam-Padenga Identity Hub receives variable name data in real-time or from batch files from individual modules. Variable name data comes into Alam-Padenga Identity Hub as it appears in its native systems (sources/modules). Alam-Padenga Identity Hub then derives new variable name data that enables it to perform fast and accurate linking. Alam-Padenga Identity Hub consists of the following components and applications:

![Diagram of AP Initiate Hub Architecture]

**Figure 8.1. AP Initiate Hub Architecture**

8.6.1.1 Identity Engine

The core Identity Engine should contain algorithms. The algorithms in the Identity Engine compare member records and produce scores that indicate which variable name records are likely to represent the same entity and the relative strength of the comparison.
8.6.1.2 **MPINET Server**

The MPINET server provides TCP/IP socket connections for the various API calls from the clients. The server provides an optimized proprietary communication protocol internally used by the Initiate SDKs and the core Identity Engine. The server is designed to handle multiple socket connections, is multithreaded and implements database connection pooling for optimum performance and fast response times.

8.7 **Clients and Applications**

8.7.1 **Initiate Enterprise Viewer**

A web-based client application should be developed using Java technology that enables the searching and retrieving of variable name member information. Developers can use Initiate enterprise Viewer to search for "localSingleUnderline"'s and other variable names records by submitting the desired criteria via a web client. Initiate Enterprise Viewer should be a read-only application, thus no information contained in the Initiate database can be altered through the application (the only exception to the read-only rule is the ability to attach notes to variable name member records).

8.7.2 **Initiate Auditor**

Initiate Auditor should be a Java-based client and can be used for resolving various tasks that require human intervention. Variable name member records are automatically linked when their scores indicate they are perfectly matched. When the scores are fairly high, but not over the auto-link threshold, Alam-Padenga Identity Hub should then generate various tasks that variable name data analysts can examine to determine whether the members should be linked. (The types of
tasks include Potential Overlays, Potential Duplicates, Potential Linkages and Review Identifiers.)

8.7.3 Alam-Padenga Identity Hub Manager

Alam-Padenga Identity Hub Manager, delivered should be bound with the base Alam-Padenga Identity Hub to provide capabilities for user management and configuration of the process. Similar to Initiate Auditor, Alam-Padenga Identity Hub Manager is a Java-based client.

8.7.4 Initiate Identity

Hub Manager should be connected to the Identity Engine using the Initiate Java SDK which internally handles the connection to the MPINET server. Reports accessing the Initiate database directly, Initiate’s reporting solution extracts variable name data from the Initiate database to create a reporting database. From this reporting database, developers and consultants can access pre-packaged reports or create custom reports.

8.8 Alam-Padenga Identity Hub basics

To help explain these concepts, let’s take a quick look at the variable name record created when “localSingleUnderline” variable name is created in the finance module.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
<td>localSingleUnderline</td>
</tr>
<tr>
<td>Variable Name Location</td>
<td>Finance &amp; Admission Modules</td>
</tr>
<tr>
<td>Variable Name Type</td>
<td>Alphabetic</td>
</tr>
<tr>
<td>Variable Name Source ID</td>
<td>FM</td>
</tr>
<tr>
<td>Variable Name Member ID</td>
<td>2245</td>
</tr>
<tr>
<td>Variable Name Status</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>

Table 8.1. Variable Name Record Properties

Using this example, we will see how Alam-Padenga Identity Hub software dynamically and intelligently matches the variable name record we just created with the information we already know about the variable name “localSingleUnderline”.

8.9 Alam-Padenga Identity Hub software

It is important to understand the composition of Alam-Padenga Identity Hub and what some of the internal attributes are. Alam-Padenga Identity Hub has three primary components. First, represented on the far left of the following picture, are the integration components. These are the Java, C++, Web Service, and .NET Application Programmer Interfaces (APIs) that Initiate Systems uses to build solutions. Second is the Identity Engine itself, which is used for processing and intelligently, linking all the identified variable name data. Third is the relational data store, which is used to store and manage Alam-Padenga Identity Hub related variable name data and records.
8.10 Supported Platforms

Alam-Padenga Identity Hub should be able to support data from many incoming data source systems like Relational Database Management Systems (RDBMS) (including Oracle™, Informix™, Microsoft SQL Server™, and IBM™ DB2), formatted flat file systems, messaging systems, and Enterprise Application Integration (EAI) systems. Additionally, Alam-Padenga Identity Hub™ should also support the storage and management of its own information in industry-standard RDBMS like Oracle, Microsoft SQL Server, Informix, and IBM DB2.

![Diagram showing components of AP Initiate Identity Hub]

Figure 8.2. Components of AP Initiate Identity Hub

8.11 Inbound Message-Based Transaction Service Processing

Typically, after a variable name source module has been identified and becomes available, information will pass from the source system to the Inbound Message-Based Transaction Service. As illustrated in the picture below, the source system (which, in our scenario, is the finance module), passes the "localSingleUnderline" variable name record to the Inbound Message-Based Transaction Service. The Inbound Message-Based Transaction Service conceptually has three parts: a reader, a process engine, and access to the Identity Engine. The main function of
the Inbound Message-Based Transaction Service is to efficiently feed the incoming variable name data to the Identity Engine via MPINET.

**Table 8.3.** 

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
<td>Inbound</td>
</tr>
<tr>
<td>Variable Name Scope</td>
<td>Financial</td>
</tr>
<tr>
<td>Variable Name Type</td>
<td>Alphabetic</td>
</tr>
<tr>
<td>Variable Name Value</td>
<td>EN</td>
</tr>
<tr>
<td>Variable Name Flags</td>
<td>None</td>
</tr>
</tbody>
</table>

**Figure 8.3. Inbound Message-Based Transaction Service Processing**

The Inbound Message-Based Transaction Service receives the incoming variable name data via the Inbound Reader. The Inbound Reader, whose main function is to receive the variable name data and acknowledge its receipt, then works in conjunction with the Inbound Message-Based Transaction Service process to parse the variable name attributes contained in the incoming variable name data (e.g., variable name, variable pointer address, variable scope, variable type, et cetera.). The attributes gathered from the incoming variable name data are then used within the Identity Engine to create the definition of a variable name member object.

**8.12 Standard Types of Member Objects**

A variable name member object is the basic unit in Alam-Padenga Identity Hub—the lowest represented information whose identity is to be established. Within Alam-Padenga Identity Hub we provide configuration templates, which define four common types of variable name member objects (pointers, Customer/Guest,
Provider, and Organization). Additionally, Alam-Padenga Identity Hub enables organizations to configure any other type of member object to properly represent their data.

8.13 Identifiers Used In Alam-Padenga Identity Hub

It is equally important to understand how Alam-Padenga Identity Hub uniquely identifies each variable name member object. Without properly and uniquely identifying each object, selecting, comparing, scoring, and linking become very difficult. To identify participating variable name objects, Alam-Padenga Identity Hub should use unique keys provided by each source module, and a set of unique keys based upon its own processing. Below are four of the common keys used to identify variable name member objects within Alam-Padenga Identity Hub.

8.13.1 Variable Member ID (VarMemIdnum)

Source modules, like the finance module’s system in our example, send data to Alam-Padenga Identity Hub. In general, these source modules uniquely identify each of their own records for easy processing and retrieval. Alam-Padenga Identity Hub captures these unique variable name record identifiers from each source system and associates it as that record’s member ID within Alam-Padenga Identity Hub. Variable Member IDs are then used within Alam-Padenga Identity Hub for processing, retrieval, and updates.

8.13.2 Variable Source ID (VarSrcCode)

Variable Source IDs are unique IDs that identify each variable name data source individually. The unique combination of variable name member ID and variable name source ID allows Initiate solutions to see which module has
provided which variable name records. This can be used for synchronization and changes to that record.

8.13.3 Variable Enterprise ID (VarEID or EntRecno)

Alam-Padenga Identity Hub assigns Variable Enterprise IDs (EID) to variable name member objects. Variable Enterprise IDs are generated as a result of the Identity Engine’s scoring and linking process. All variable name members sharing a Variable Enterprise ID represent an entity relationship (see Entity below) for that record. As an example, if there are two variable name member records for “localSingleUnderline” having the same Variable Enterprise ID with an entity type of Identity, then Alam-Padenga Identity Hub has determined that these two records represent the same member object.

8.13.4 Variable Member Record Number (VarMEMRECNO)

Alam-Padenga Identity Hub assigns this number to the variable name member’s record. This value is unique for each record. Since the IDs are not static, when variable name members change, the Variable Enterprise ID can also change. Any variable name member can always be known by its VariableSourceID/VariableMemberID, and any linkage set can be known by any one of the set’s members. For example, when a new member’s record is added to the system, the Variable Enterprise ID for the variable name member will be the same as the VarMemRecno. If another record is added for the same variable name member in a different source (module), these two records will be linked and both records will be assigned the same Variable Enterprise ID (indicating there is a linkage), but the VarMEMRECNO will not change.
8.14 Entity Types

An entity is defined as "something that exists as a particular and discrete unit." In terms of identity management, an entity is the logical relationship between two or more variable name records. Entities are represented in the Initiate environment as variable name records sharing a common Variable Enterprise ID. An entity can also be called a "linkage set." There can be an unlimited number of variable name records in an entity or linkage set. An entity type allows for distinction between the way variable name members are viewed and linked within Alam-Padenga Identity Hub process model. Each entity type has a specific algorithm configuration. The two most common entity types are Identity and Household.

8.14.1 Identity

For the Identity entity, variable name records from the various source modules that belong to the same variable name member are identified by the matching algorithm and linked with a common Variable Enterprise ID. When developers wish to link variable name member records of individual variable members, such as "localSingleUnderline", it can use the Identity entity. In this case, the algorithms compare the records to determine if the record belongs to an individual module. For instance, all records of "localSingleUnderline" are identified in all of the modules and linked; all records of "localAdditionSum" are identified and linked, and so on.

8.14.2 Household

Another type of entity is Household, where the entire variable name members' information is linked together based on a location-oriented algorithm. For example, consider "localSingleUnderline"'s family: localSingleUnderline, localDoubleUnderline, localBoldUnderline, localAsterixUnderline, and
localDashUnderline. "localSingleUnderline" has a variable name record in the finance module, and localDoubleUnderline. localBoldUnderline and localAsterixUnderline have their variable name records in the admission module. They will all be linked as one entity known as Household. In this case the algorithms would not only check if the variable name record belongs to one individual for the Identity entity, but would also check if the record belongs within the same household as another variable name member. (This would be useful, for instance, when records of all the family members might be required by developers to upgrade all local underlining variable names.) Variable Enterprise IDs are not changed when variable name members are linked within a household entity.

8.14.3 Group

In normal processing, a variable name record can only belong to a single entity within an entity type. Group entity functionality enables records to have multiple entity variable name record numbers (varEntRecno) within a single entity type. A member of a group entity must match all variable name members of that entity set above the autolink threshold.

8.14.4 Identity Engine Process Overview

Consider the variable name record for "localSingleUnderline" from the incoming finance module has been received by the Inbound Message-Based Transaction Service and is now being passed to the Identity Engine for processing.
As shown in the picture above, the Identity Engine is logically divided into three processes:

1. Construct core variable name member data,
2. Derivation process, and
3. Matching process.

The Inbound Message-Based Transaction Service passes variable name data to the Identity Engine. The Identity Engine then constructs and stores the variable name member object. It then:

- Standardizes the variable name data from the source variable name record
- Intelligently derives new variable name data from the incoming variable name record
- Selects other variable name records already in Alam-Padenga Identity Hub that might have the same information
• Compares the information attribute-by-attribute from each variable name record
• Generates a score for the incoming variable name records
• Dynamically links variable name records that meet the business threshold

The configurable algorithms in the Identity Engine create the derivation variable name data, compare variable name member records and produce scores that indicate which variable name records are likely to represent the same entity and the relative strength of the comparison.

8.15 Constructing Core Variable Member Data
As our variable name record from the finance module (see table below) enters the Identity Engine, the Identity Engine first constructs the core variable name member object used to identify and compare the variable name records of the module. Constructing core variable name member object involves storing the parsed variable name data from the Inbound Message-Based Transaction Service.

8.15.1 Verticalization
The construction of core variable name member data is actually a unique “verticalization” process, where all relevant attributes are stored as separate rows, and possibly in separate tables, in the Initiate database. Consider our example of the variable name record created for “localSingleUnderline” in the finance module: Remember that from the above information, the Inbound Message-Based Transaction Service has created unique packets of information to be handed to the “construct core variable name member” process. The construct core variable name member process writes the variable name data to the associated tables in the database.
Table 8.2. Variable Name Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
<td>localSingleUnderline</td>
</tr>
<tr>
<td>Variable Name Location</td>
<td>Finance &amp; Admission Modules</td>
</tr>
<tr>
<td>Variable Name Type</td>
<td>Alphabetic</td>
</tr>
<tr>
<td>Variable Name Source ID</td>
<td>FM</td>
</tr>
<tr>
<td>Variable Name Member ID</td>
<td>2245</td>
</tr>
<tr>
<td>Variable Name Status</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>

8.15.2 Benefits of Verticalization

Alam-Padenga Identity Hub employs several unique child tables, each to store individual attributes of “localSingleUnderline” (such as variable name, variable type, and variable source) for improved computational speed, additional processing, and express access. Attributes are stored in the various database tables such as variable name member address table, variable name member name table, variable name member type, et cetera. Thus, unlike a typical data structure where each member has one row of data, the information is stored in separate tables.

Each variable name member object can have more than one row in each of these tables. This is the foundation for Alam-Padenga Identity Hub as it makes the variable name data model so robust. This is useful in cases where one of the sources has an extra location or an extra aliases as these can be easily stored in the variable name member location and the variable name member attributes tables.
Alam-Padenga Identity Hub stores and manages variable name data within industry standard databases. Initiate's schema should be designed to be flexible and efficient when selecting, comparing, scoring, and linking the vast amounts of variable name data across an application package. The Initiate solution should also allow developers to exploit many of the high availability and scalability features natively provided by each of these vendors (i.e. Oracle Real Application Clusters or IBM High Availability Clusters Multi-Processing).

8.15.3 Derivation Process

Once the core variable name member data is created and written, the next step within the Identity Engine is the derivation process (depicted below). The derivation process within the Identity Engine can actually be broken up into three logical sub-processes:

1. Standardization of the incoming variable name data,
2. Bucketing variable name data (or segregating new variable name data entities into specialized buckets), and
3. Comparison variable name data (or the extraction of variable name data from the incoming source for comparison purposes).

![Figure 8.5. Derivation Process](image-url)
8.15.4 Standardization

Standardization is the resolution of certain attributes within the data to a common format. For example, the variable name “local_Single_Underline” could be standardized to “LOCALSINGLEUNDERLINE” after removing the underscores and changing all characters to uppercase.

8.15.5 Bucketing

After the standardization process is complete, the derivation process performs a bucketing process on the variable name data. This is where groups of attributes, which will form the various buckets that are identified during the initial configuration of Alam-Padenga Identity Hub, are grouped together. Typically, a solution requires about five to seven buckets per variable name member. In each bucket one or more attributes can be involved. For example, the buckets may be [variable name], [variable Type and Size] and [variable location]. Buckets help in narrowing the search by identifying a group of candidate variable name records who share some of the same bucketing information. “localSingleUnderline”’s record in the finance module might yield the following bucket information: “localSingleUnderline”, word, 50, FinMod(Finance Module).

8.15.6 Comparison Data

The final step in the derivation process is the generation of additional comparison variable name data. All of the attributes that are used in the comparison process are extracted from the core variable name data and stored separately in the derived variable name data layer. Now we have two types of variable name data, which constitute the derived variable name data layer. First is the bucket variable name data, which is used in the candidate selection process, and second is the candidate comparison variable name data, used when comparing two specific variable name members.
8.15.7 Matching process

Once Alam-Padenga Identity Hub has completed the derivation process, we can now begin to apply different meaningful matching and comparison algorithms to the variable name data to determine if the incoming variable name record is or is not related to other variable name records already processed in Initiate. The algorithms take full advantage of the derived bucketed and derived comparison variable name data created in the derivation process above.

![Figure 8.6. Matching Process](image)

The key to Alam-Padenga Identity Hub is in how it compares and achieves a finer definition of how two or more variable name records may be related. Other solutions stop at the highest level of comparison and can lead to many “false positives” because of the way they store the variable name data and the limited
matching capability. Alam-Padenga Identity Hub is different, as can be seen in the picture above, because its matching process actually involves four logical steps:

1. Selecting a group of candidate variable name records.
2. A comparison process, which compares each of the candidate variable name records to the incoming record attribute by attribute.
3. Scoring the incoming variable name record.
4. Linking variable name records that meet the established business criteria.

8.15.8 Selecting Records

This portion of the process involves the selections of candidate variable name records from the variable name records already received and scored within Alam-Padenga Identity Hub. Candidate selection is the collection of known variable name records that will be further analyzed in the comparison process. As stated earlier, the Identity Engine calculated a series of values useful for comparison for our incoming variable name record—"local," "Single", "localSingle", "SingleUnderline", "Underline", word,50 and "FinMod" (Finance Module)—and stored them in the database. The process uses each of these values to compare against the values of the already existing variable name member's objects in the database, and creates a list of possible candidates who share the bucketing information. Below is an example of the list of potential candidate records that might be passed to the comparison sub process.
8.15.9 Comparison process

The two logical steps in the comparison process are compare and score. These two steps differentiate Alam-Padenga Identity Hub from all other approaches.

8.15.10 Compare

This step actually compares the individual attributes gathered at derivation time, attribute-by-attribute, against the records already in Alam-Padenga Identity Hub. Functions that determine the degree of similarity between the attribute values should be incorporated. The results from these comparisons are then passed to the scoring process.

8.15.11 Score

The results of the comparison are given a numerical score to indicate the likelihood that the variable name records refer to the same variable name member. Scores can be negative (implying non-relationship) or positive (implying relationship). The greater the score, the more similar the variable name members are to each other. Alam-Padenga Identity Hub utilizes the score of the candidates to indicate the next
processing step. Three defined thresholds determine the actions the Identity Engine based on the resulting score when comparing variable name members. The threshold for each solution can be adjusted based on the business case that needs to be solved.

8.15.12 Thresholds

Thresholds are scoring levels set to determine how variable name records will be managed by during Initiate phase. There are three thresholds which are discussed below:

8.15.12.1 Auto-link threshold

As depicted in the image below by the red triangles to the right of the auto-link (AL) threshold line, when the value of the score of two variable name records is above the auto-link threshold, Alam-Padenga Identity Hub automatically links these records. This value indicates that the particular variable name member object already exists in the database and therefore the variable name member is automatically grouped/linked with other variable name member records and assigned a common VarEnterpriseID. (Note that the data is not merged, but is only linked.)

![Figure 8.7. Scoring Threshold](233)
8.15.12.2  Task Threshold (also known as clerical review)

Indicated in the picture above is the manual review band. The value of this band is below the auto-link threshold but above the business acceptable lowest boundary or task threshold. When the calculated score is between task threshold and auto-link threshold, the record is thought to be a high enough degree of similarity to warrant manual review. The resulting action is to trigger an "issue or task record" for review. Through the use of Initiate Auditor, a developer can determine whether these variable name records are duplicate records or they should be linked.

8.15.12.3  Overlay Threshold

The value of overlay threshold (OVL) is configurable and tends to be a negative value. If an update to a variable name member record, when compared to the prior information for the variable name member, score is below the overlay threshold value, then the variable name data comprising the individual variable name member has been changed to such a degree that it implies the record ID may be representing two different variable name objects. This also triggers an issue variable name record for a manual check, which then can be handled through the Initiate Auditor phase. This usually occurs when there is an update and the existing information is completely different from the new information, suggesting that there is an overlay of the record with another variable name member's information. This is represented by the red triangles below the lower threshold value. Alam-Padenga Identity Hub can thus be configured such that:

- No tasks are created and only auto-links will occur
- Only tasks are created and no auto-links will occur
- Neither tasks nor auto-links will occur
- Both tasks and auto-links will occur
8.15.12.4 Link

The final step in the matching process is the actual linkage step. Linkages are two or more records grouped together by a common Variable Enterprise ID. An unlimited number of records can be included in a linkage set. As discussed previously, there are two types of linkages:

1. Automated link – The link generated automatically by Alam-Padenga Identity Hub
2. Manual link – The link created manually by an end-user while resolving a task

A linkage occurs when:

- Two variable name records share a common Variable Enterprise ID
- The score, created by a weight-based comparison of the attributes, determines an automatic linkage
- Linkages can be instantiated manually, regardless of score

Referring back to our example, the incoming record for “localSingleUnderline” needed to be compared against three candidate variable name records already in Alam-Padenga Identity Hub. The attributes of each variable name record are passed into the comparison process (an actual comparison process would usually include many more attributes than this example) and each variable name record is scored. Based on the limits of this example, we are going to infer that the only logical variable name record from the three variable name records related to “localSingleUnderline” is the variable name record for “localDoubleUnderline” (a household relationship). If our threshold evaluation placed this record above the auto-link threshold then the variable name record for “localDoubleUnderline” would automatically link to the incoming record for “localSingleUnderline”. The
other two records—"localAdditionSum" and "localMultiplicationProduct"—probably scored low and may require further evaluation. Alam-Padenga Identity Hub criteria and threshold evaluation would either create an "issue variable name record" or not link these records entirely.

8.15.13 Managed Task Queues
As mentioned earlier, Alam-Padenga Identity Hub does not assume that because a variable name record fails to score above the auto-link threshold that it should not be linked. A task may be created, which facilitates user review and possible resolution. A task is a unit of work, represented by "issue variable name records." There are four task types.

8.15.14 Tasks

8.15.14.1 Potential Overlay – An overlay occurs when one variable name member’s information overrides the information of another, different, variable name member record within the same source (thus, the same Source ID is shared by two variable name records). An indication of this may be if a variable name member record within a source has its attribute data change radically.

<table>
<thead>
<tr>
<th>Existing Record</th>
<th>Record Update</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attribute</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>Variable Name</td>
<td>localAdditionSum</td>
</tr>
<tr>
<td>Variable Name Type</td>
<td>Alphabetic</td>
</tr>
<tr>
<td>Variable Name Source</td>
<td>Finance Module</td>
</tr>
<tr>
<td>Variable Name SourceID</td>
<td>101</td>
</tr>
<tr>
<td>Variable Name MemberID</td>
<td>2245</td>
</tr>
</tbody>
</table>

8.4. Existing Record 8.5 Record Update

8.15.14.2 Potential Linkage – If two variable name records from different sources have a score between the clerical review and auto-link thresholds, a
Potential Linkage task is created. The variable name record in the finance module and the variable name record in the admission module for "localSingleUnderline" score close, but the variable name records did not score above the auto-link threshold.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
<td>localSingleUnderline</td>
</tr>
<tr>
<td>Variable Name Type</td>
<td>Alphabet</td>
</tr>
<tr>
<td>Variable Name Source</td>
<td>Finance Module</td>
</tr>
<tr>
<td>Variable Name SourceID</td>
<td>101</td>
</tr>
<tr>
<td>Variable Name MemberID</td>
<td>2248</td>
</tr>
</tbody>
</table>

Existing Record

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
<td>UniqueNumber</td>
</tr>
<tr>
<td>Variable Name Type</td>
<td>Admission Module</td>
</tr>
<tr>
<td>Variable Name Source</td>
<td>Admission Module</td>
</tr>
<tr>
<td>Variable Name SourceID</td>
<td>201</td>
</tr>
<tr>
<td>Variable Name MemberID</td>
<td>3149</td>
</tr>
</tbody>
</table>

Record Update

8.6 Existing Record

8.7 Record Update

8.15.14.3 Review Identifier – Two or more variable name records that contain the same unique identifier (for example a variable name number) requires a user to manually link them or correct the identifier if one variable name member has the identifier in error. In addition to the four task types discussed above, there are two system tasks:

8.15.14.4 HasShadow and PreMerge: Developers reviewing variable name records and tasks will not see these two types in Initiate applications as they are used primarily by the Message-Based Transaction Services to trigger an internal system action.

8.15.15 Task Workflow

Alam-Padenga Identity Hub can place notifications of tasks into managed work queues. Initiate Auditor is a sophisticated; application designed specifically for efficient resolution of task managed queues and associated reporting. The resolution of a task by the application developer determines the state of the task. These resolution states can be configured for each installation. Some example resolutions could be:
- Not the same person
- Merged – survivor
- Deferred – supervisor review. If a developer is not able to resolve a task based on the information available, they can change the status of the task to "Deferred."

This helps in identifying the tasks that were reviewed but not resolved. It does not, however, have any impact on the task. The task rows may be deleted; the variable name member may be promoted into a linkage or dropped out of the task set (see Dynamic nature of tasks below). If a task is marked as resolved, then it is automatically deleted and removed from a managed queue.

8.16 Dynamic Nature of Tasks

The dynamic nature of tasks is reflected in a number of things. When a task variable name record is selected, the task is dynamically formed based on the real-time comparison of that variable name member’s data to others at that point in time. Task records also can be re-formed based on new or changed information— the associated score could increase or decrease, the issue record could be removed, or the issue record could be promoted to higher task type in the hierarchy. When issue variable name records are resolved based on developer input, they are removed from the queue.

The variable name record, or criteria, that the Initiate phase uses to compare against other variable name records (candidates) is called the trigger variable name member or trigger variable name record (i.e., the variable name record triggering the comparison). When a task search returns results, the trigger variable name member typically has a higher score than the other variable name records. The
score is higher based on the trigger variable name record being compared against itself during the comparison process. The other variable name records returned are issued comparison scores based upon their comparison against the trigger variable name record.

A variable name record is also considered to be the trigger variable name member if an update to that variable name member causes a cross-match which results in task creation. For example, variable name record C is assigned a Potential Duplicate task in comparison with variable name records A and B. Any changes to C will affect the task status. Hence C is said to be the trigger variable name record. This capability where a task could get automatically deleted or changed based on the update to the trigger variable name record makes the tasks dynamic. In short, a trigger variable name record is a member variable name record that has a variable name record in the task table. A, B and C may all have task variable name records, and therefore would all be considered trigger variable name records.

8.17 Task Hierarchy

Although a variable name record may have numerous variable name data issues, Initiate phase only allows a variable name record to be part of a single task at one time. The assignment is based on a task type hierarchy—some variable name data issues are more important than others and should be resolved first. This hierarchy—from highest to lowest—is:

- Potential Overlay
- Potential Duplicate
- Potential Linkage
- Review Identifier
Occasionally, updates to variable name member data can cause a new, more serious, data issue. When this occurs, the variable name record is promoted to the higher task type. Likewise, updates can cause the variable name data issue to be removed, thus removing the task.

8.17.1 Record Promotion

The tasks as mentioned above are listed in the order of importance (most important to least important). Whenever there is a change that involves a new issue with an existing trigger variable name record, the variable name record is assigned the more important of the two task types. The information about the tasks is stored in the following tables:

8.17.2 Rules

Rules are pair-based and are set between two variable name members. Rules can be used to overrule the decisions made by Alam-Padenga Identity Hub during comparison. For example, there may be two variable name records of duplicates and they might compare very high, but an administrator can review them and set as a rule that the two variable names are never the same (non-identity rule). Therefore, even if there is an update on the variable name records, these particular variable name records will never be linked or even put in the review queue. Another kind of rule is the identity rule, where a developer manually links two variable name records. The rule then says that the two variable name records are always the same, regardless of future variable name data changes.

8.17.3 Versioning – The Lifecycle of an Attribute

Each of the attribute tables contains a variable name record status column called “recstat”, which determines the state of the variable name record. The possible values for “recstat” are:
- **Active**: This status indicates that this value is the last known value to be true. (Usually corresponds to last update.)

- **Inactive**: This status indicates that this value is the previously known value to be true.

- **Deleted**: This status indicates that this value has been logically deleted and will not be used any longer for comparison purposes, but is still stored to maintain history.

- **Shadow**: This status indicates that this is the updated value pending approval from a source. After receiving an update message from the source that confirms the change, this flag turns to “Active” state.