

# KMx Enterprise: Integration Overview for Member Account Synchronization and Single Signon

---

KMx Enterprise includes two api's for integrating user accounts with an external directory of employee or other user information and to allow Kmx to participate in a singlesignon authentication installation within an enterprise environment.

**SOAP User Import API:** The insert, update, and deactivation or deletion of user accounts is accomplished via the SOAP protocol which exposes a method for posting a batch of 1 – 500 user records at a time. The batch consists of a single xml payload, posted as a string parameter to the SOAP method call with instructions and information to act on the associated user records. SOAP UserImport API includes the WSDL discovery language so that developer tool kits can quickly generate and implement the appropriate method calls. The api is usually located on a server with KMx installed at <http://server.domain.com/services/userimport.asmx> The schema defining the xml to be passed to the userimport method can be located on a server with KMx installed at <http://server.domain.com/services/xsd/userimport.xsd>

**Flat File User Import:** Customers wishing to submit a flat text file to KMx instead of using the SOAP API for user import and synchronization may instead submit a file via ftp to a location specified by KMSI. A windows service will pickup the file and process it for import. The flat file format specification is documented in Appendix 2 of this overview.

**Single Signon API (SSO):** The Single Signon API or SSO uses the HTTP GET protocol to pass a user token, an organization token, a timestamp and a hash code to the api. This allows for a user to authenticate against a portal, or via an LDAP directory in an intranet, or some other vehicle external to KMx then have their browser session directed to KMx already authenticated without being challenged again for a username and password. The SSO API is typically located on the KMx server at <http://server.domain.com/dotnet/application/singlesignon.aspx> A description of how the hash code is calculated to insure that the request was authenticated on the trusted server (LDAP, portal, etc) is included in Appendix A of this document.

**Separate API's:** These two functions are separated to offer KMSI customers the most flexible options available to when planning their deployment and integration strategies. Here are three common scenarios:

- 1) Batch calls to UserImport with no SSO implementation. A batch process synchronizes user accounts in KMx with an authoritative source such an HRIS system, but users still access KMx via the native KMx authentication interface, and are challenge for a username and password, each time they log in.
- 2) Run time calls to UserImport then SSO. This scenario works well for integrating KMx with a consumer facing portal. A user logs in to the portal, requests a resource on the Kmx platform, and this immediately initiates a call both api's

- UserImport first, the SSO. The user account is either created or updated in kmx and then the user is immediately re-directed via SSO to KMx to begin his learning activity. It should be noted that this scenario never deactivates a user account in kmx. It relies on user action to cause an update, and when the user stops using the portal, no further action occurs to cause any further change. This usually isn't a problem in a consumer facing scenario, but can be in an enterprise environment where the training materials are proprietary and continued access by terminated employees represents a security threat or is otherwise undesirable.
- 3) Batch UserImport for account creation and SSO for authentication – In an enterprise where user profile data may be split among many systems, a batch process can periodically query these systems (usually once per day) and compile a profile for KMx then call the UserImport api to update the appropriate profile(s). Such implementations should be sensitive to recognizing accounts that should be deactivated in KMx and include periodic instructions for this as well (ie include terminated in employees in the batch with instructions to delete their accounts, or write a separate batch to address this need.). Such an enterprise may also have an LDAP directory or other common authentication scheme and wish to use this to provide improved security and convenience to employees. In this scenario, the SSO event may still include a call to update the affected users' profile if desired.

### **Example Application**

KMSI has developed a sample web application that demonstrates the implementation of calls to both API's. It is written in Microsoft ASP.net and VB.net v1.1. It assumes that the trusted source for both of account profile data to pass to the UserImport API and for authentication is Microsoft Active Directory and that the KMx web server is deployed where it has access to the same AD domain(s) as those used to authenticate the users.

It is intended as an example only. Error trapping, logging and other production ready elements of a robust application have been omitted in the interests of keeping the sample code as concise as possible.

The source code will be made available free of charge to customers to who have a valid KMx Enterprise license and elements of the example may be incorporated in to the final integration at the sole discretion and risk of the customer.

The example uses a single web page authenticated under AD to query the AD for user profile details, initiate a batch update of that one user's account, then redirect the user's browser to the SSO api.

The example call to the UserImport API should be instructive even to those planning batch update implementations as it includes classes to compile the xml payload that this api requires regardless of the operating context of the client application calling the api.

### **Major elements of the example application:**

There are two web pages included with the sample application:

**ShowLoggedInUser.aspx** – will simply display the first name, last name, full name, email, objectGUID, and phone number, username, and a recommended password for use with kmx, for the logged in user as authenticated under AD. It is instructive for testing and use of ActiveDirectoryUser class, but should not be incorporated in to any final deployment as it exposes information that might constitute a violation of local security policies.

**KMxSignonWithAD.aspx** – This is the actual application sample. It queries the ActiveDirectoryUser Class to get profile details about the logged in user, calls the UserImport api to update the profile, then redirects the user's browser to the SSO api.

**Web.config** – stores authentication and location parameters necessary to complete the api calls.

**Web Reference/KMxSoapAPI** – this class was built with Microsoft Visual Studio code generation wizard from the WSDL for the UserImport api and includes the methods to actually invoke the API.

**ActiveDirectoryUser.vb Class File** – Provides a very simple set of methods and properties to model a simple user profile from an active directory user account. This class could be extended to query other data sources and compile a comprehensive profile for your integration.

**CryptographicUtilities.vb Class File** –Provides utility methods for calculating the MD5 hash code required by the SSO api.

**KMxUserImportPackager.vb Class File** – provides methods to compile the xml required by the UserImport API.

**DocumentsAndReferences Directory** – Contains a copy of this summary document, a data dictionary for the elements of the xml payload to be compiled in to the UserImport API, and a copy of the XSD schema file KMx uses to validate the UserImport data prior to processing.

**Data Dictionary** – The spreadsheet “KMxMember Data Dictionary.xls” summarizes all of the elements in the member profile that can potentially be imported. It includes a definition for each field, a summary of the data format and value restrictions, and it's multiplicity value, indicating if it is required, optional or how many times it may occur. Most fields are self explanatory however, the directory\_services\_id and supervisor\_directory\_services\_id merit additional discussion here.

**directory\_services\_id** – This is the foreign key that uniquely identifies an individual within your organization. It must be unique for each person in your

organization and should not be expected to change over time. A value like an employee id assigned by an HRIS system or a Global Unique Identifier (GUID) assigned by a Microsoft Active Directory implementation or other unique value is a good choice. Last names, email addresses, full or partial social security numbers, or some combination of these tend to be less desirable as they do periodically change over time.

**supervisor\_directory\_services\_id** – this field represents the `directory_services_id` of the individual's immediate supervisor. It must be selected and defined the same way the `directory_services_id` above is defined, but the value provided will represent the individual's immediate supervisor rather than the individual himself. Providing this information is optional. Doing so allows for reports to be generated that model reporting structures and organizational spans of control. Without, such reporting and analysis will not be possible. It should be provided only if it is being drawn for an application or database such as an HRIS where the information is being regularly maintained so that it is current and accurate.

## Appendix 1: KMx Enterprise - MD5 Hash Pass-through Authentication

### Overview

*What is MD5 hash?*

MD5 hash is a one-way function that takes bytes as input and output bytes that represent a “fingerprint” of the input.

*What does MD5 hash accomplish for our authentication?*

It ensures that only your website (portal, intranet, etc) can send requests to authenticate a user to the KMx Platform. A user could not type the URL into the browser and be successfully authenticated into the external vendor’s website.

*How does MD5 hash work?*

The input of the MD5 hash function will be the concatenated string of parameter values you are passing to the KMx Platform plus a “Shared Secret”. The output of the function will be the hash value (digest) that you will pass to the KMx Platform as an additional parameter.

*What prevents a user from reusing the same URL?*

One of the parameters that will be included in the hash input string is a timestamp. This allows the KMx Platform to disregard any requests with timestamps older than a certain time (eg. 5 minutes)

*How is MD5 hash implemented?*

Many programming environments include an MD5 hash function in their standard libraries. Other languages have extensions that support MD5 hash.

To help in your research of MD5 hash on your platform, consider visiting the following sites.

General Info:

<http://www.ietf.org/rfc/rfc1321.txt>

Cold Fusion:

<http://www.cflib.org/udf.cfm?ID=35>

Active Server Pages:

<http://www.freevbcode.com/ShowCode.Asp?ID=2366>

<http://www.sloppycode.net/asp-components/>

<http://www.hotscripts.com/Detailed/5098.html>

<http://www.hotscripts.com/Detailed/3381.html>

<http://www.search4scripts.com/search.cgi?k=encryption>

JavaScript

<http://pajhome.org.uk/crypt/md5/>

### **Requirements**

A mechanism for passing an authenticated user from a generic portal or other authenticated website to the KMx platform in a safe and secure manor:

- The website must be authenticated. The message must be authenticated to verify the origin of the message.
- The website must be secure. No unauthorized parties should be able to access the user information.
- The website must be resistant to a relay attack. No unauthorized parties should be able to relay the user information to gain access.
- The website must be expandable. The message must be able to carry arbitrary data.

### **Terms**

- *Source Site* – the site that produces the message.
- *Destination Site* – the site that receives and processes the message. (The KMx Platform in this case).
- *Message* – the data that is passed between the source site and the destination site, including raw data and the digest.
- *Raw* – the original data enclosed in the message.
- *Digest* – The output of the MD5 hash algorithm
- *Shared Secret* – A phrase agreed upon by the source and destination that is not transmitted with the message.

### **Authentication**

The destination site must authenticate the message from the source site. This specification requires the following algorithm based on keyed MD5 hash authentication.

- The source site and destination site will agree on a secret phrase.
- The source site assembles the raw data that needs to be passed.
- The source site appends the secret phrase to the raw data and produces the digest using MD5 hashing algorithm.
- The source site produces the message with the original data and the digest, but without the secret phrase.
- The source site sends the message to the destination site.
- When the destination site receives the message, it extracts the raw data and digest from the message.

- The destination site appends the secret phrase to the raw data and produces the digest using MD5 hashing algorithm.
- The resulted digest is compared with the digest in the message. If they are the same, then the message is authenticated.
- The destination site processes the raw data.

### **Secure Transport**

When the user clicks the link to the destination site, the browser will be redirected to the destination site URL with the message passed as a query string. The name is *data* and the *value* is the message properly encoded. For example: *https://destination\_site:port/start\_link?data=message*.

The transport protocol must be secure to protect the data. It is recommended to use HTTPS protocol, or any other secure transport protocol.

### **Message Format**

The message will be transmitted using HTTP request parameters. The raw data (the input of the MD5 hash function) will be a concatenated string of the *values* of the request parameters plus a shared secret. The name of the parameters will not be included in the raw data.

### **Parameters**

Descriptive name	HTTP parameter name	Description	Max Length	Concatenation order
Profile ID	profileId	The source site profile identifier string for the user. This must be a unique ID and must be the same ID used as <i>directory_services_id</i> when synchronizing user Accounts via the KMx SOAP API (see KMx SOAP for details).	40	1
Time Stamp	timestamp	The time stamp is a long integer number of the millisecond from the year 1970 (epoch time)	N/A	2
MD5 Hash	hash	The digest that results from running the MD5 hash function over a concatenated string of the above parameters plus the shared secret. Please see below note regarding MD5 hash format.	N/A	N/A
Access Key	Accesskey	Integer value assigned by KMx. Necessary to distinguish between	N/A	N/A

		multiple organizational accounts on KMx ASP. KMSI will assist with assigning the specific value		
--	--	---	--	--

**MD5 Hash:**

The input value of the MD5 hash algorithm is computed by concatenating the parameter values. The “concatenation order” column specifies the order of concatenation. The shared secret must be concatenated to the end of the parameter values before calling the function.

Typically MD5 hash algorithms output binary data. Since this output does not conform to legal HTTP request parameters characters sets, the MD5 Hash binary output should be encoded as an ASCII string. The simplest encoding is hexadecimal encoding. Another encoding type is Base64, if deemed necessary. Some implementations of the MD5 algorithms return ASCII data rather than binary data; in this case, the implicit encoding type must be discovered so the destination site can use the same encoding for the comparison. (In other words, implementations that return a string value, implicitly encode the return value in Hex or Base64).

Example:

Profile ID: **320001**

Date: Wed **Aug 18 12:44:58 EDT 2004**

Timestamp: **1092847498202**

Shared Secret: **g9yMzVwK**

<https://kxmplatormhost.domain.com/dotnet/application/singlesignon.aspx?profileId=320001&timestamp=1092847498202&hash=b895b2f8f0ca021d15fe1b1226dee5e3&accesskey=37>

## **Appendix 2 – Flat file specification.**

Customers wishing to take advantage of the user import api of KMx, but who do not wish to compile input data as XML, may alternatively use this file specification to prepare a text file and submit it to KMx via ftp. The KMx background service will then convert the file to xml for submission to the SOAP api.

### **Separator Characters:**

**Record Separator:** Each row of the text file will be assumed to represent one record. Rows shall be separated by a carriage return/line feed pair (ascii characters 13 and 10 respectively).

**Field Separator:** The vertical line character, sometimes referred to as the “pipe” character (ascii value 124) will be used as a field separator between fields within a record.

### **First Row: Authentication Record**

The first row of the text file will contain the authentication record. The authentication record consists of 3 fields and represents the user credentials of the system administrator under whose profile the import will be processed.

**Field 1 – username:** this is the username of a KMx system administrator under whose permission the import is processed. In most cases it is desirable to create a user account in KMx with System Administrator privileges specifically for this purpose rather than using the same account as an actual administrator. This field is required and must appear exactly once in the authentication record. Data type: string with minimum length = 1 and maximum length = 50.

**Field 2 – password:** this is the password of the same KMx system administrator represented by the username above. This field is required and must appear exactly once in the authentication record. Data Type: string with minimum length = 1 and maximum length = 50

**Field 3 – access key:** this value will be provided by KMSI and is used to segregate which customer organization the file represents among customer’s of KMSI’s hosted service. This field is required and must appear exactly once in the authentication record. Data Type: integer.

### **Second and Subsequent Rows: Batch Member Record**

After the authentication record, each subsequent row is assumed to represent one member (user) record.

The member record for a flat text file shall consist of 47 fields as defined in the attached data dictionary. The fields must appear in the order listed in the data dictionary and with the value restrictions indicated in the data dictionary.

### **Blank Values vs. “No Value Submitted” –**

Of the 40 fields defined in the data dictionary only two have a multiplicity of “Exactly 1” meaning they must be specified in the record submitted. These are the command to execute upon the record, and the `directory_services_id` of the record to operate on. All others are optional.

In the XML syntax native to KMx, an optional value may be omitted by simply not listing it in the record. When a value is omitted, it is handled by KMx according to one of two rules:

- On Insert – the KMx default value for this field is applied.

- On Update – the current value for this field is left in tact, regardless of how this value was set (previous import, update by an administrator, update by the user, etc.).

If it is intended that instead the XML explicitly set the value in question to a blank string (where permitted by data type) then the optional field is included in the record with the blank value explicitly set.

Since a flat file requires every field in the record to be included every time and the field cannot be simply omitted, KMx has established a special value to distinguish between these cases for a flat file import.

The value “NoValueSubmitted” (case sensitive, do not include quotes) may be used for any field in the data dictionary that is listed for multiplicity “0 or 1”. It may be submitted regardless of data type to indicate that the optional field is not addressed by the input file and should instead be addressed by KMx according to the same rules as an XML input with the field in question omitted, that is:

- On Insert – the KMx default value for this field is applied.

- On Update – the current value for this field is left in tact, regardless of how this value was set (previous import, update by an administrator, update by the user, etc.).

To explicitly set a string value to a blank string (where permitted by data type) customers may submit the field with no data between the separators.

A sample flat file is included with the KMx sample integration application.

## Appendix 3 - Implementing Organizational Hierarchy in KMx Enterprise v3.3

---

**Definitions** – For purposes of this appendix document the following definitions apply.

Organization – The superior most organizational unit recognized by KMx. It is modeled by the organization table in the KMx data system and all objects in the KMx system schema are subordinate to it.

Company – a sub unit of people within an organization, such as a department, division, subsidiary, team, etc. Companies may be defined to be subordinate to one another.

**Concept of Operations** - KMx Enterprise v3.3 will include the ability for enterprise customers to model their organizational hierarchy and structure within the members table holding the user accounts. This will be accomplished by adding a metadata field to each member record called member\_company\_mask. This field will represent the company to which a member belongs. It will be structured in such a way that it also represents that company's position in the overall organization hierarchy. This extends the current ability of KMx to group members by company and to include any companies subordinate to a particular company.

Customers who choose to populate and maintain this structure will in turn be able to group reports according to the company hierarchy and delegate some administrative tasks with access privileges grouped according the same hierarchy.

**Mask structure** – the member\_company\_mask (or simply the “mask”) will be constructed by as follows.

Every company to be tracked in this manner is assigned its own alphanumeric designator by the customer. This can be any convenient designation desired such as an organizational number assigned by an HRIS system.

An individual's mask is the concatenated value from right to left of all company designators for companies superior to his own, plus his own company designator, followed by wild cards to pad the mask to a consistent length for all members.

The most superior organizational company, usually the CEO's office, is assumed to be the head of the organization and needs no company designator in the mask. The CEO and his executive staff can be assigned a mask that is simply a string of the wildcard padded to the consistent length for all of the organization's masks. (Please note that this default mask for the CEO will still require its own company record. See the implementation section for more details.)

For example, if the World Wide Broadcast Monopoly consists of three divisions, NBC, ABC and CBS, and each division president has multiple department managers reporting

to him, who in turn have multiple team leaders reporting to them. This hierarchy can be modeled with a 9 character mask as follows.

CEO: mask: \_\_\_\_\_ (9 wildcard characters)

CEO's admin assistant mask: \_\_\_\_\_ (9 wildcard characters)

NBC Division President mask: NBC\_\_\_\_\_

NBC Pres' admin assistant: mask: NBC\_\_\_\_\_

Dept Mgr 005 at NBC mask: NBC005\_\_\_\_

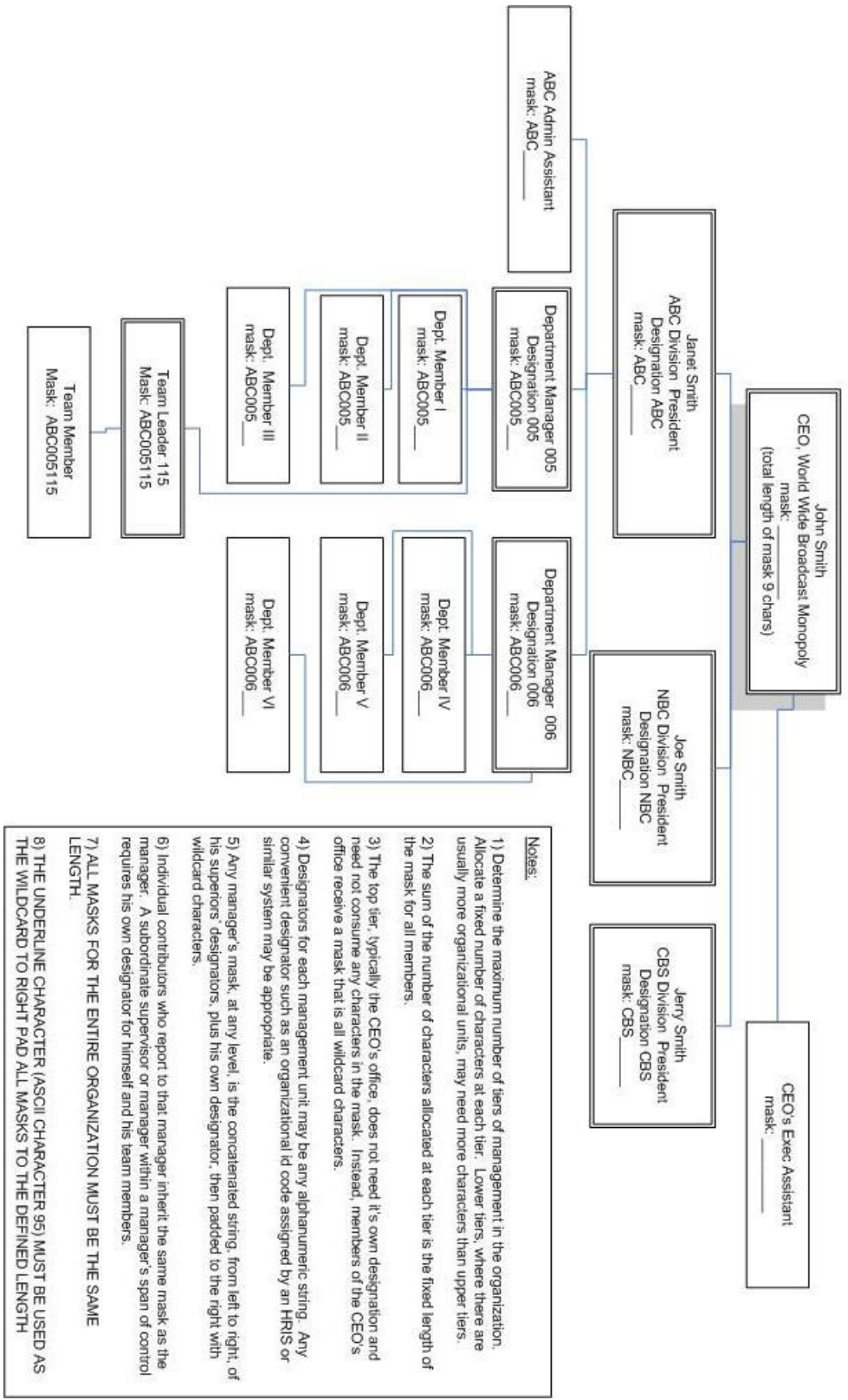
Dept 005 Members at NBC mask: NBC005\_\_\_\_

Team leader 115 at Dept 005 at NBC mask: NBC005115

Team 115 members mask: NBC005115

# KMx Enterprise v3.3 Company Mask Example Values

Figure 1 illustrates this example more completely.



- Notes:**
- 1) Determine the maximum number of tiers of management in the organization. Allocate a fixed number of characters at each tier. Lower tiers, where there are usually more organizational units, may need more characters than upper tiers.
  - 2) The sum of the number of characters allocated at each tier is the fixed length of the mask for all members.
  - 3) The top tier, typically the CEO's office, does not need its own designation and need not consume any characters in the mask. Instead, members of the CEO's office receive a mask that is all wildcard characters.
  - 4) Designators for each management unit may be any alphanumeric string. Any convenient designator such as an organizational id code assigned by an HRIS or similar system may be appropriate.
  - 5) Any manager's mask, at any level, is the concatenated string, from left to right, of his superiors' designators, plus his own designator, then padded to the right with wildcard characters.
  - 6) Individual contributors who report to that manager inherit the same mask as the manager. A subordinate supervisor or manager within a manager's span of control requires his own designator for himself and his team members.
  - 7) ALL MASKS FOR THE ENTIRE ORGANIZATION MUST BE THE SAME LENGTH.
  - 8) THE UNDERLINE CHARACTER (ASCII CHARACTER 95) MUST BE USED AS THE WILDCARD TO RIGHT PAD ALL MASKS TO THE DEFINED LENGTH.

The following guidelines should be followed when planning the structure and implementation of the mask:

- 1) Determine the maximum number of tiers of management in the organization. Allocate a fixed number of characters at each tier. Lower tiers, where there are usually more organizational units, may need more characters than upper tiers.
- 2) The sum of the number of characters allocated at each tier is the fixed length of the mask for all members. Max length 50 characters.
- 3) The top tier, typically the CEO's office, does not need its own designation and need not consume any characters in the mask. Instead, members of the CEO's office receive a mask that is all wildcard characters.
- 4) Designators for each management unit may be any alphanumeric string. Any convenient designator such as an organizational id code assigned by an HRIS or similar system may be appropriate.
- 5) Any manager's mask, at any level, is the concatenated string, from left to right, of his superiors' designators, plus his own designator, then padded to the right with wildcard characters.
- 6) Individual contributors who report to that manager inherit the same mask as the manager. A subordinate supervisor or manager within a manager's span of control requires his own designator for himself and his team members.
- 7) ALL MASKS FOR THE ENTIRE ORGANIZATION MUST BE THE SAME LENGTH.
- 8) THE UNDERLINE CHARACTER (ASCII CHARACTER 95) MUST BE USED AS THE WILDCARD TO RIGHT PAD ALL MASKS TO THE DEFINED LENGTH
- 9) Once the mask pattern is defined, be sure to register all possible mask values as company records in the company table of KMx. This includes a company record for the CEO's mask (all wild cards). This is necessary to support administrative filtering of members and other KMx objects according to the organizational hierarchy in KMx v3.3 and future releases.

**Implementation Goals** - This approach is implemented with the following goals in mind:

- 1) **Functionality** - To deliver to KMx customers the functionality necessary to administer and report on a members' training and learning activity according to the organization hierarchy.
- 2) **Flexibility** – This approach can be used to model any organizational structure and to maintain that structure with information provided from any number of information sources external to KMx such as SAP, Lawson, Peoplesoft, Oracle HR, or other source of organizational information a customer may select for integration.
- 3) **Performance** – The mask structure is consistent with KMx's high performance and scalability architecture. It pre-indexes the hierarchy and avoids the need for recursive logic to compile reports or calculate admin access privileges at run time.

### **Implementation** –

**Database Schema** - KMx v3.3 will extend the database schema as follows to support this new functionality:

New Field: members.member\_company\_mask nvarchar(50)

New Field: company.company\_mask nvarchar(50)

**SOAP API** – The SOAP API for KMx v3.3 will be extended to expose member\_company\_mask as part of the member profile. This will provide KMx customers the ability to maintain the mask value for individual members.

**KMx Admin Interface** – The KMx administrative interface will be extended to expose the company.company\_mask field. Optionally customers may also expose the members.member\_company\_mask field if they wish to access this field directly. This may be quite useful during testing of a new integration. However, customers are cautioned to consider whether those with Personnel Admin rights to KMx should be editing this field manually once in production. The default configuration of KMx v3.3 will not expose members.member\_company\_mask to the admin interface.

**Company Table** – Once the mask pattern is determined and all possible mask values are calculated, they should be registered in the company table of KMx via the administrative interface as follows: Administer Personnel → Insert New Company. During initial set up, it may be desirable to have a qualified database administrator load the list of possible values directly to the company table. Be sure to populate the company\_name and company\_mask fields. Other contact fields about the company are optional. Pricing related fields in the company table represent deprecated functionality and are supported for backward compatibility with earlier versions of KMx only.

As new companies are created within the organization or if the organization is restructured, the company table must be updated accordingly.

**Company Administrators' menu** - Company administrators access their administrative rights through the student interface. This capability was first introduced in KMx v3.0 and will continue in v3.3 and beyond. Those designated as company administrators (designated by member\_auth = 10) who currently have access to only those members directly assigned to the same company, will now inherit the same access to all subordinate companies in addition to their own. Any member of the company can be designated a company administrator by setting member\_auth = 10 via the SOAP api, or by using the "Review Member Authorization Level" dialog in the KMx System Administration menu.