HP System Dictionary XL
General Reference Vol 1

HP 3000 MPE/iX Computer Systems

Edition 3

Manufacturing Part Number: 32256-90004
E1287

U.S.A. December 1987

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).

Acknowledgments

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

1 Introduction

Overview

This chapter provides a brief description of System Dictionary/XL and the operating environment required for the product. It also provides information about the programming languages and subsystems that System Dictionary supports.

Data Dictionaries

Just as an ordinary dictionary is a collection of definitions of words, a data dictionary is a collection of definitions and descriptions of data that resides on a computer system. In a dictionary, the smallest unit of information is a word, while in a data dictionary the smallest unit of information is a **data element**.

You can compare a data dictionary to the file card system in a public library. Each card in the file contains a description of a book in the library, and lists the title, author, publisher, location in the library, etc. of the book. The card itself is not the book, but only represents and describes it.

Like the file cards, a data dictionary does not contain the data itself, but contains **metadata** --data about data. This metadata can be descriptions and definitions of various kinds. It can describe such things as:

Data: Names and definitions of data elements

Data Relationships: How data is related to other data

Data Responsibility: Who is responsible for what data

Organizational Structure: The information flow, who uses the data

Location Information: Where files, programs, and reports reside

Security Information: Who has access to what data

A typical example of a piece of metadata is a data element called "SSN", which represents a piece of data--a social security number. The social security number *itself* does not reside in the dictionary, but a description of that piece of data does. For instance, the data dictionary might tell you the name of the data element, the storage length, the display or output length, the type of data (numeric or character), its sign if the element is numeric, which database or program that data resides in, and possibly which departments in the organization use and maintain that data.

The data dictionary, therefore, serves many purposes. You can use it as a quick directory to the information that resides on a computer system--where to go to get pieces of data. You can also, however, use it as one of the primary means for ensuring consistency of data definitions and preventing data redundancy. This means that programmers and developers may be required to check the data dictionary for data elements that already exist on their system before they design a new program.

Therefore, if a data element already exists on the system describing a social security number (for example, "SSN"), the dictionary reports this information and does not allow the programmer to add a new element with the same name. This helps an organization to save time in program development by using data definitions that already exist. It also saves data storage space by preventing data redundancy and helps to standardize data definitions within an organization.

For a more detailed introduction to what data dictionaries are and how an organization can use a data dictionary, see the HP primer entitled *Managing Your Information Network: A Data Dictionary Primer.*

System Dictionary Description

HP System Dictionary/XL is a flexible and extendible data dictionary that is designed to be a central information resource and index to programs, users, input forms, and network configuration on a 900 Series HP 3000 system. You can customize and localize this dictionary to meet the needs of your data resources.

The information contained in HP System Dictionary/XL defines, describes, and identifies the data in your resources, and the relationships between that data.

HP System Dictionary/XL software is divided into two products:

- HP System Dictionary/XL, which consists of the core set, the intrinsics, and eight utilities: SDMAIN (the user interface), SDINIT, SDCONV, SDDBD, SDDBC, SDVPD, SDUPGRAD, and SDUTIL.
- The HP System Dictionary/XL COBOL Definition Extractor Utility (SDCDE). This utility, though sold as a separate product, requires System Dictionary to be already installed, in order to run.

Features

The Entity-Relationship Model The Entity-Relationship (E-R) model consists of entities, relationships between entities, and attributes that define and characterize entities and relationships.

Extensibility and Customization The models and applications described in a dictionary will grow. To handle this growth, System Dictionary provides the ability to extend the dictionary, by adding entity types, relationship types, relationship classes, attributes, and type-attribute associations.

N-ary Relationships You may need to express a relationship between more than two entities. System Dictionary supports relationships between six entities. This ability allows greater consistency of information when renaming or deleting definitions.

Dictionary Domains Although most of the definitions in a dictionary are meant to be commonly shared by all of you, many of you need separate name spaces, or separate partitions for different applications. HP System Dictionary allows you to define entities and relationships within a local domain, which you may use to separate applications or duplicate names.

Version Control You may have separate dictionaries to model your test and production environments. System Dictionary allows you to define multiple versions of an entity or relationship within a single dictionary.

Security HP System Dictionary will be central to your network processing environment, often containing sensitive information. It therefore provides a security scheme that allows you to access the dictionary only if you are authorized, and controls your access to dictionary data.

Utilities System Dictionary includes several utility programs, as described below:

- SDINIT initializes and re-initializes the dictionary.
- SDMAIN is a user interface that creates, maintains, and retrieves dictionary components.
- SDCONV is a program that converts and loads data from Dictionary/V to System Dictionary.
- SDDBD is a program that loads information about a TurboIMAGE/V database structure from an TurboIMAGE/V root file into System Dictionary.
- SDDBC creates TurboIMAGE/V schemas and root files, using System Dictionary data as the source.
- SDVPD is a program that loads information about VPLUS forms from a VPLUS forms file into System Dictionary.
- SDUPGRAD is a program that automates the inclusion of new, HP-provided definitions into the core set.
- SDUTIL is a general purpose utility program which has the following capabilities:

1. Allows users to merge parts of a dictionary into another dictionary, providing better dictionary standardization and data resource management.
2. Allows users to create a compiled dictionary, using a master dictionary as the source, providing users with a fast-access, read-only dictionary, which can save them a significant amount of time.
3. Allows users to rename a dictionary.
4. Allows users to purge a dictionary.

- SDCDE, though closely linked with System Dictionary, is sold as a separate product. This utility generates COBOL II data definitions and copylibs from System Dictionary definitions.

Benefits

HP System Dictionary provides the following benefits, that can help you create and manage efficient, cost-effective data resources.

- Minimizes data redundancy
- Supports system auditing and reporting
- Assures data integrity
- Improves system maintainability
- Tracks and controls data resources
- Reduces program development time
- Facilitates the standardization of naming conventions
- Enforces security rules
- Provides for analyzing the system impact of data and relationship changes

2 Dictionary Concepts

Overview

This chapter provides a basic description of System Dictionary. More details are provided in chapters 3, 4, and 5.

Entity-Relationship Model

System Dictionary uses a simple but powerful entity-relationship (E-R) model in which the central components are entities, relationships between entities, and attributes that define and characterize entities and relationships. You can use the System Dictionary E-R model to describe the real-world objects of an information network, and to define logical connections between these objects.

Entities and Relationships

Entities and relationships are the definitions that are stored in and retrieved from the dictionary. Entities are dictionary definitions that refer to objects in the real world of an information network. An entity has attribute values that further define the real-world object or that describe the entity itself. A relationship is an ordered list of entities. Relationships express logical connections between real-world objects. Most relationships involve two entities and are called binary relationships. System Dictionary also supports N-ary relationships involving three to six entities. Relationships have attribute values that in some cases describe the dictionary definition itself and in other cases describe one of the entities in the relationship. Therefore, you can use relationship attribute values to document an entity in a particular context or usage.

Structures

The System Dictionary E-R model provides a set of structures that support the creation and retrieval of entities and relationships. Entity types are structures that define, categorize, and qualify entities. For example, IMAGE-DATABASE is a System Dictionary entity type that defines entities that describe IMAGE databases. Relationship types are structures that define, categorize, and qualify relationships. IMAGE-DATABASE contains IMAGE-DATASET is a System Dictionary relationship type that defines relationships between an IMAGE database and data set.

Attributes are structures that define the attribute values of entities and relationships. IMAGE-DATASET-TYPE is a System Dictionary attribute associated with the IMAGE-DATASET entity type. CAPACITY is an attribute associated with the IMAGE-DATABASE contains IMAGE-DATASET relationship type.

In System Dictionary, each entity type and relationship type has an attribute list. Attributes are associated with the entity type or relationship type by a structural component called a type-attribute association. When an attribute is associated with an entity type or relationship type, each entity or relationship of that type must have a value for the attribute.

System Dictionary supports six attribute types. Five of the types are fixed-length:

- Alias--A 32-byte character value
- Boolean--A 1-byte true or false value
- Character--A 1 to 255-byte, inclusive, alphanumeric value
- Floating--A 32-bit or 64-bit floating-point format
- Integer--A 16-bit or 32-bit binary value

The remaining attribute type is variable. Alias and variable types are free-floating, in that they are never assigned to entity types or relationship types, but you can assign values for them to any entity or relationship. An alias attribute contains an alias value for an entity or relationship. An attribute of type variable, also referred to as a variable-length attribute, has no maximum length, and normally stores descriptions, edit masks, default values, long names, and other variable-length values. Since not every entity or relationship needs alias and variable-length attributes, System Dictionary sets aside storage space for them only if you explicitly assign them. When you delete an alias or variable-length attribute, the storage space is released.

A relationship class is a name, like contains or uses that describes that action or connection of a relationship. In System Dictionary, a relationship type belongs to a relationship class. The relationship types RECORD contains ELEMENT and FORM contains ELEMENT, for example, belong to the contains relationship class. In some cases, a relationship class serves as a qualifier for a relationship type. For example, System Dictionary has three relationship types that involve element pairs:

ELEMENT contains ELEMENT

ELEMENT redefines ELEMENT

ELEMENT references ELEMENT

The first type creates relationships between parent and child entities. The second type indicates that two elements share common storage in a program. The third type documents an element that references another element, as in a Pascal type reference. These ELEMENT ELEMENT relationship types are qualified by a relationship class.

Extensibility

System Dictionary provides a built-in set of structures called the core set. The core set includes entity types, relationship types, relationship classes, attributes, and type-attribute associations. You can extend your dictionaries by creating new entity types, relationship types, relationship classes, attributes, or type-attribute associations. You are also allowed to make limited changes to the core set, such as changing the external names of structures. The extensibility feature allows you to customize the dictionary. It also allows Hewlett-Packard to localize System Dictionary to your native language, and to update the core set as new subsystems are added to MPE.

Domains and Versions

In System Dictionary, you can divide a single physical dictionary into multiple, logically separate domains. A domain is a name space that contains entities and relationships. You typically use domains to keep the entity and relationship definitions of one application group separate from others. You can also use domains to avoid naming conflicts. Every System Dictionary has a built-in domain called the common domain. The common domain is always present and is always public. You can add your own domains, called local domains, and assign them a sensitivity level of public or private.

You can further subdivide domains into named partitions called versions. Versions allow you to designate one set of entities and relationships in a domain as the current production version, and other sets as test or archival versions. Unlike domains, which are considered separate name spaces, you should think of versions as complete copies of the entities and relationships in a domain. In a version of test status, you can add or delete entities and relationships, or change their attribute values. You cannot modify an archival version, and you usually keep this version for historical purposes. Also, you cannot modify a production version. System Dictionary allows only one production version in a given domain. The version feature allows you to maintain a stable production version while you experiment with other versions. It also allows you to reactivate an archival version if it becomes necessary to return to a previously used set of definitions.

An important function of a data dictionary is to ensure the standardization and integrity of the definitions you use in an information system. The domain and version features allow you to experiment with new definitions and maintain separate sets of definitions. System Dictionary has security features that prevent unauthorized users from creating domains and versions, and that prevent one user from creating new versions of an entity or relationship that another user owns.

Dictionary Security

System Dictionary provides a comprehensive security scheme consisting of scopes, sensitivity levels, and scope associations. A scope is a dictionary user name with a password and a set of capabilities called scope rights. The System Dictionary scope rights determine whether you can create or merely read entities and relationships, extend the dictionary structure, create scopes and obtain information about dictionary security, and create domains and versions.

Scopes own entities, relationships, structures, domains, versions, and other scopes. Some operations require you to be the owner of the dictionary object on which the requested operation is to be performed. Every dictionary has a built-in scope named CORESET that owns the entity types, relationship types, and other structures of the core set. When you create a dictionary, you name and assign a password to a Dictionary Administrator (DA) scope. The DA scope automatically has all scope rights plus other capabilities not available to other scopes.

Each entity and relationship has a sensitivity level that determines whether other scopes can read or modify it. Domains also have sensitivity levels that designate them as public or private.

If an entity or relationship has a sensitivity level of read or private, its scope-owner can grant a higher level of access to a selected scope by creating a scope association.

Naming Considerations

All names used in System Dictionary are handled exactly the same. This subsection provides general information and rules for handling names within System Dictionary.

Syntax

The syntax for all dictionary names is the same, with the exception of entity naming, which is discussed in Chapter 3 under Special Entity Naming. All names must be 32 characters, left justified, and right blank filled. No blanks are allowed between characters, however. All lowercase letters are upshifted. You may use all alphanumeric and special characters in a name EXCEPT the following.

. System Dictionary Restricted Characters

. , ; : ! " ( ) < > ^ =

When you create or access an object, you can supply any combination of upper or lowercase characters in the name, but System Dictionary always upshifts the name. Thus orders, Orders, and ORDERS all refer to the same object.

Name Sets

Each name in System Dictionary must be unique within its set of names. The following sets of names exist in the dictionary:

- domain names
- version names within a domain
- entity type names

- relationship class names
- attribute names
- scope names
- entity occurrence names within a domain within an entity type

Each of the above sets is divided into two parts, internal names and external names, which are described below. Each part is considered a set of names.

Internal and External Names

Every dictionary definition, whether a structural component (for example, entity type) or an entity occurrence, has both an internal name and an external name associated with it. The internal name, which can never change, is intended for use by software products used with the dictionary that rely on given names for identification purposes. The external name, which is fully customizable and localizable, is intended for end users or those accessing the dictionary through a user interface facility.

This flexible scheme allows software developers to extend the structure of the dictionary or add occurrences using internal names of a personal nature (for example, ISV-RESOURCE entity type) that are less likely to conflict with existing names. You can then use the external names to make the extensions more friendly. For example, if a RESOURCE entity type did not exist, then ISV-RESOURCE could be externally named RESOURCE.

System Dictionary requires only one name to create a definition in the dictionary. You can use that name as both the internal and external name provided that no conflict occurs with any existing names. For example, an end user creates a domain MY-DOMAIN and adds it to the dictionary. That name will become both the internal and external name. However, if that name conflicts with an existing internal or existing external domain name then the creation is not completed. The internal and external names are considered to be part of different sets of names. The same name can exist in both sets, but it must be unique within each set.

When opening the dictionary, you must specify whether you are using the internal or external name set during the current dictionary session. During that session, with the exception of creation operations where you can supply one or two names, you can use only one name type.

NOTE In the future, Hewlett Packard will prefix all names of objects it adds to the core set with "HP". To avoid potential name conflicts, do not create any entity types, relationship types, relationship classes, attributes, scopes, or domains, prefixed with "HP".

Internal Numbering

System Dictionary associates an internal number to each component within System Dictionary. The responsibility for these numbers is completely under the control of System Dictionary. The number value System Dictionary assigns to a particular component does not necessarily follow any set pattern. The SDMAIN program and the System Dictionary intrinsics use internal numbering.

Dictionary Control Operations

Dictionary control operations consist of the functions that set global controls on the dictionary, and are described separately below.

Initializing/Reinitializing the Dictionary

You can initiate System Dictionary only through the program SDINIT. The procedure for running this program is explained in the first part of the HP System Dictionary/XL General Reference Manual, Volume

Opening the Dictionary

Opening the dictionary is the procedure that defines a new access path to a dictionary you specify. You establish an access path with the information you specify in the parameters of the SDOpen intrinsic or the DEFINE command in SDMAIN. This information includes the dictionary open mode, scope, name mode, domain, and version that you are using.

After you establish an access path, you may modify it by using the SDMAIN DEFINE command or the appropriate dictionary intrinsic to switch the current name mode, scope, domain, or version. This procedure takes less time and uses fewer system resources than closing and re-opening the dictionary. This is especially true from the Exclusive Customization mode, as any changes made to the dictionary structure would be automatically incorporated at that time through the restructuring process. Closing the dictionary and calling SDOpen or specifying the DEFINE command again requires an allocation of system resources, which can degrade the performance of System Dictionary.

It is also possible to define multiple access paths to a dictionary within the same session, that is, you may call SDOpen repeatedly without closing previous OPENs. This may be useful, for example, for copying definitions from one domain to another. Multiple OPENs are not generally recommended, however, as serious problems can occur if they are not done correctly.

Remote Dictionary Access

System Dictionary includes a feature that allows remote dictionaries. This capability allows you to create, access, and maintain dictionaries on systems located at remote sites without having to actually be there. A company, then, can create and maintain standard dictionaries in its offices worldwide from a central location, likely the corporate offices. You can also access the remote dictionaries locally, and modify them to include local definitions as needed, but you will most likely use them as "read only" dictionaries, implementing company standards and definitions throughout its offices.

For more information on how to use remote dictionaries, see the DEFINE command of the HP System Dictionary/XL SDMAIN Reference Manual or the SDOpen intrinsic of the HP System Dictionary/XL Intrinsics Reference Manual.

Compiled Dictionary

System Dictionary includes two types of dictionaries which containmetadata:

- Master Dictionaries. A master dictionary consists of a TurboIMAGE database that can be accessed by all System Dictionary intrinsics and SDMAIN commands. Because of its complexity, however, the response time while using master dictionaries may be inadequate for some subsystems, especially those that must read from the dictionary at run-time.
- Compiled Dictionaries. A compiled dictionary contains metadata extracted from a master dictionary. Like a compiled program, a compiled dictionary cannot be modified. A compiled dictionary is therefore a read-only dictionary that can be accessed by those System Dictionary intrinsics and commands that only read dictionary metadata.

Compiled dictionaries provide faster dictionary read access and are intended to be used by subsystems and applications that need only to read the metadata. A compiled dictionary is less complex than a master dictionary and is compacted into one or more flat files. Therefore, it requires less disc storage space and provides a more efficient means of transporting dictionary data to other groups and accounts in the same system, or other systems in a distributed network. For more information about compiled dictionaries, see the SDUTIL utility of the HP System Dictionary/XL General Reference Manual, Volume 2.

Merging Dictionary

The utility SDUTIL provides System Dictionary users with the capability to selectively merge certain dictionary data into appropriate areas of the same or other dictionaries as follows:

- Dictionary structure definitions can be merged into the structure of another dictionary.
- Occurrences from a version in a dictionary can be merged into another version in the same or another dictionary.
- Security definitions from one dictionary can be merged into the security scheme in another dictionary.

SDUTIL allows users to preview the results of the merge operation and provides users with information about conflicts that will occur if the merge is allowed to continue. The user can then decide whether or not to do the actual merge operation. Note that it is also possible to merge dictionary data and compile it during the same session. For more information about merging dictionaries, see the SDUTIL utility of the HP System Dictionary/XL General Reference Manual, Volume 2.

3 Dictionary Architecture

Overview

The architecture of System Dictionary includes several major components (the E-R model and its components, the core set, domains, versions, security, and the intrinsics) which are all interrelated and work together to provide the benefits listed in Chapter 1. With the exception of the intrinsics, these components are discussed individually in this and the following chapters.

This chapter provides a description of the Entity-Relationship Model, the structure on which System Dictionary is based. It also includes information on the individual components that make up this model, and tells how to use them. They are:

- Entities
- Relationships
- Entity types
- Relationship types
- Relationship classes
- Attributes
- Type-attribute associations

The main purpose of a data dictionary is to provide a mechanism for creating and accessing entities and relationships. Entities and relationships are the definitions that are stored in the dictionary. The other components of the E-R model (entity types, relationship types, attributes, and so on) are structures that support the creation and retrieval of entity and relationship definitions.

The Entity-Relationship Model

System Dictionary is based on a structure called an **Entity-Relationship** model. The theoretical model is composed of **entities**, that represent pieces of real-world data in an information network, and **relationships** between entities. This model is general enough that it can describe most, if not all, of the information processing done on a computer network.

A simple example of the E-R model might represent a data base called COMPANY, which contains a file called PERSONNEL. In this example, the entities are COMPANY and PERSONNEL, and the relationship between the two is **contains**. A standard illustration for this example is shown in Figure 3-1 below.

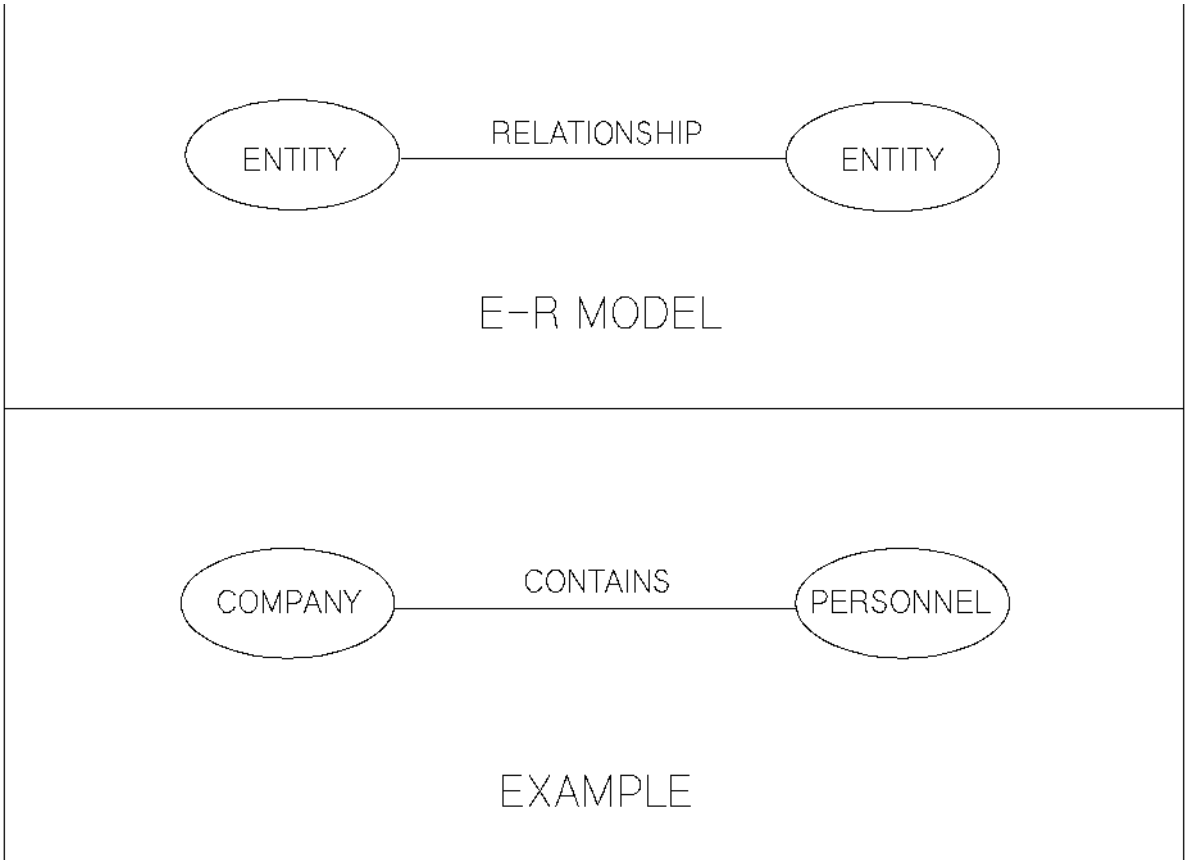


Figure 3-1. E-R Model

Within System Dictionary, the E-R model has been enhanced and also includes **entity types, relationship types, relationship classes,** and **attributes**. These are definitions within the **structure** of System Dictionary, and they support the creation, maintenance, and retrieval of the entities and relationships, which comprise the bulk of the definitions within the dictionary. The components of the E-R model are described on the following pages. The E-R structure is illustrated in Figure 3-2.

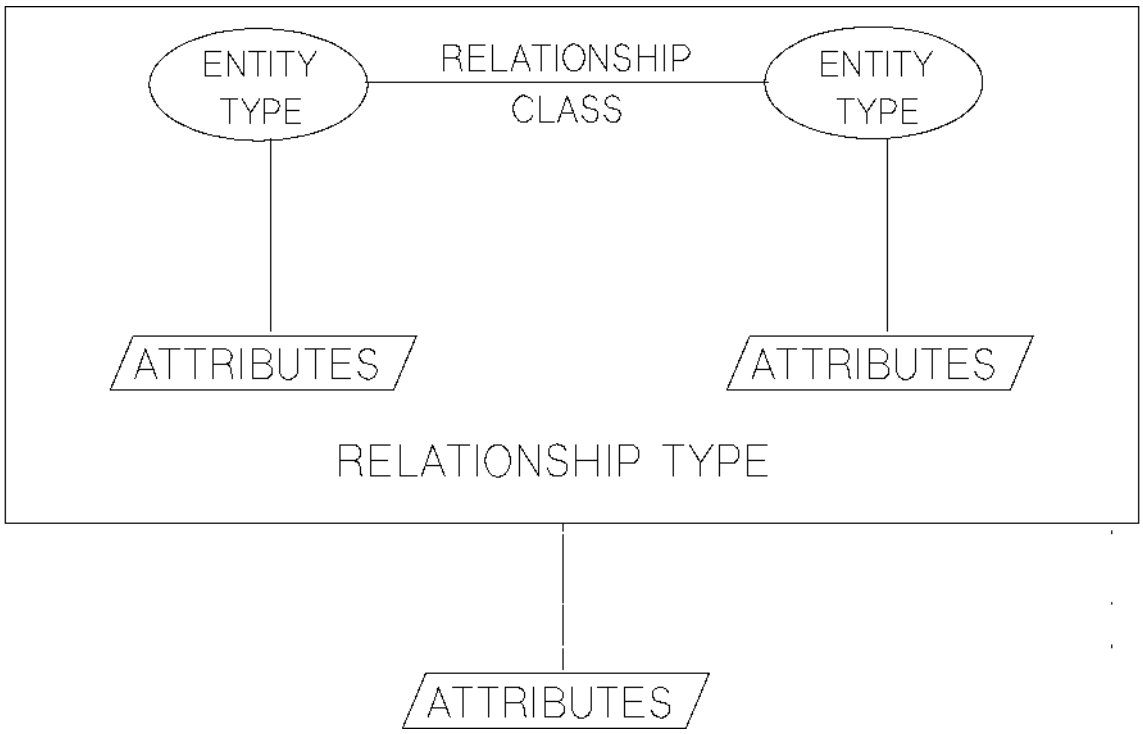


Figure 3-2. System Dictionary E-R Structure

Entities

Entities are dictionary definitions that refer to tangible objects in an information network. In System Dictionary, an entity definition has three parts:

- An entity *name* --usually the name of a real object.
- The entity *type*, a dictionary category to which the entity belongs.
- *Attribute values* that further define and characterize the entity.

An entity is therefore a unique pairing of an entity name with an entity type. For example, to create a dictionary definition of an IMAGE data base named ORDERS, you would need to create an entity named ORDERS of entity type IMAGE-DATABASE. To define a VPLUS form named CUSTOMERS, you would need to create an entity named CUSTOMERS of type FORM. If there is also an IMAGE data set named CUSTOMERS, you would need to define an entity named CUSTOMERS of type IMAGE-DATASET. The dictionary would then contain two CUSTOMERS entities, but they would be unique when qualified by entity type FORM or IMAGE-DATASET.

Special Entity Naming

Entity names have the same restrictions as discussed in Chapter 2 under the Syntax discussion. Along with those restrictions there are two reserved entity names. The two reserved entity names in System Dictionary are the slash (/) and the question mark (?). These entity names have a special meaning in creating or accessing relationships. You can use the slash name in a relationship to indicate a null entity. You can use the question mark name in relationship retrieval to indicate that any entity name is acceptable in a given position. Entity names may *include* slashes and question marks. The restriction is that you cannot use them *as* entity names except in the relationship contexts just described. For example, you cannot create an entity named ?, but you can create entities named ORDER?NUMBER or A/B.

Specifying Entity Types

Entity types are dictionary categories that you must specify when you create, delete, modify, or retrieve an entity. When you create or access an entity, you must specify an entity type that already exists in the dictionary. If the entity type you specify does not exist, the access fails. System Dictionary has a built in set of entity types, called *core set* entity types. You can also add your own *extended set* of entity types. Any entity type in the core set or extended set is available to all users of the dictionary. The core set and extended set are discussed in Chapter 6 of this manual.

An example of entities and entity types is illustrated in Figure 3-3. The top half of the figure shows the entity types (as ellipses) and the attributes of those entity types (as boxes). The lower half shows specific entity occurrences (CORP and R&D) of the respective entity types, and the attribute values of the attributes associated with them.

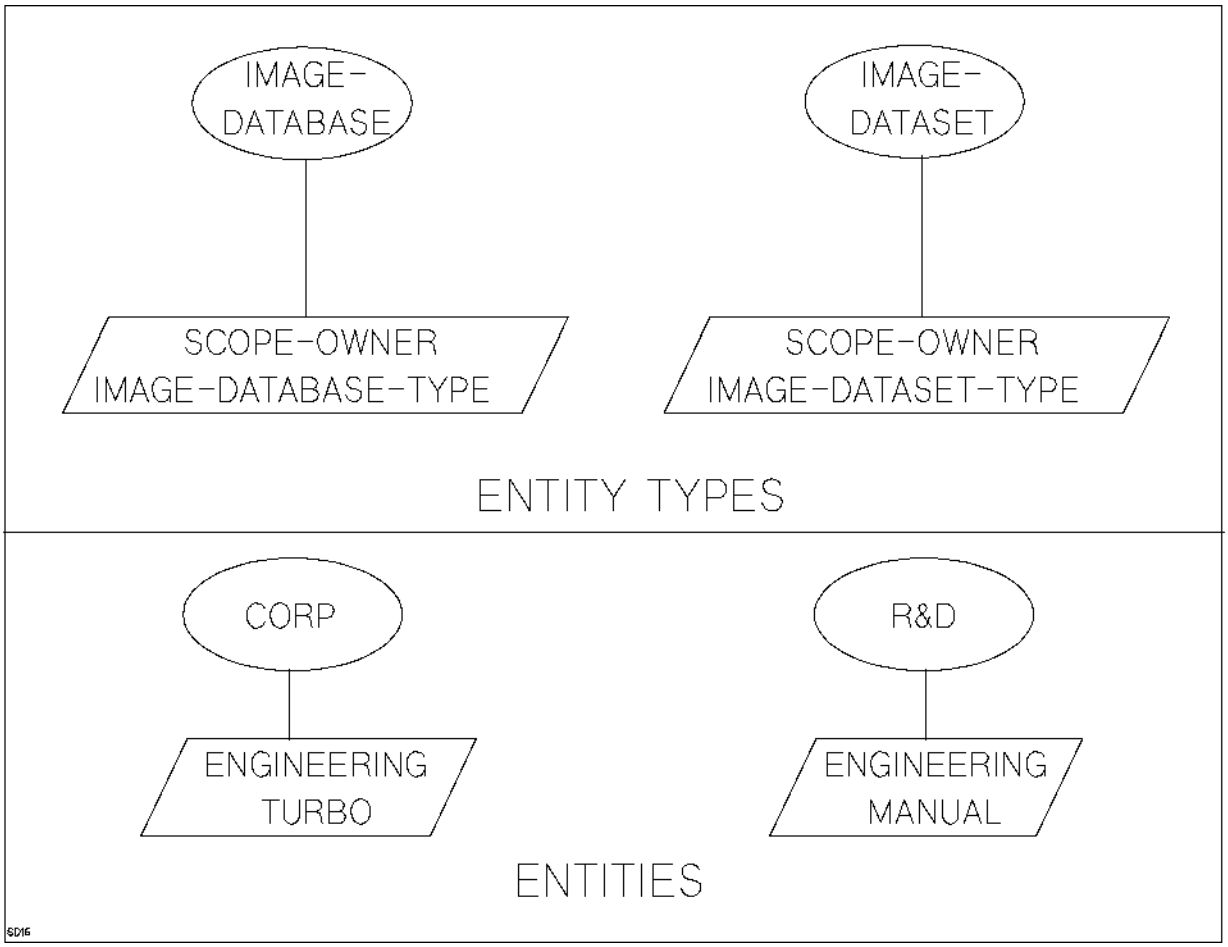


Figure 3-3. Entity Types vs. Entities

Entity Attribute Values

Entities have attribute values that define and characterize the entity definition itself (when it was created in the dictionary, the owner, etc.) or that describe the entity's tangible object. Table 3-1 shows some sample entities of the entity type IMAGE-DATASET. Each entity in the table refers to a data set in an IMAGE data base. In the first column of the table are the entity names. The other columns contain values for the attributes SCOPE-OWNER, DATE-CREATED, and IMAGE-DATASET-TYPE. The SCOPE-

OWNER and DATE-CREATED attributes describe the entity definitions themselves--who owns them and when they were created in the dictionary. The IMAGE-DATASET-TYPE attribute describes each entity's real-world object, a particular IMAGE data set. In the CUSTOMERS entity, for example, the IMAGE-DATASET-TYPE attribute indicates that the CUSTOMERS data set is a manual master.

Table 1: Entities of type IMAGE-DATASET

IMAGE-DATASET	SCOPE-OWNER	DATE-CREATED	IMAGE-DATASET-TYPE
CUSTOMERS	DA	5/20/86	MANUAL
SALES	RON	4/18/86	RELATION
BURGERS	KELLY	4/25/86	DETAIL
CHEESE-CODES	SAM	7/21/86	AUTOMATIC

Every entity of type IMAGE-DATASET must have a value for each of these attributes (plus other attributes not shown in the abbreviated table). You can think of an entity type as defining an imaginary "table" of entity information. Each "row" in the table corresponds to an entity, including its name and attribute values. The entity type has a list of associated attributes, and each attribute has a "column" in the table. When you create an entity, a row is added to the table, and you must specify a value in each attribute "column" (or allow the attribute value to be defaulted).

When you retrieve an entity, you can request all or part of the conceptual "table" by specifying a complete or partial attribute list. Only the values corresponding to the attributes you specify are returned. The complete attribute list of the IMAGE-DATASET entity type is shown in Table 3-2. When you retrieve an entity of type IMAGE-DATASET, you can request a value for any of these attributes. System Dictionary supports attributes of type alias and variable-length (described later in this chapter). These attributes are "free-floating" in that they are never associated with an entity type, but you can assign values for these attributes to any entity. Not every entity needs alias or variable-length attributes, so the as-needed method of attribute assignment helps conserve space in the dictionary.

Table 2: Core-Set Attribute List Examples

Entity Type	Core-Set Attribute List	
ELEMENT	DATE-CREATED	COUNT
	DATE-CHANGED	DECIMAL
	SCOPE-OWNER	DISPLAY-LENGTH
	SCOPE-CHANGED	ELEMENT-TYPE
	SENSITIVITY	JUSTIFY
	ID-NUMBER	SIGN
	BLANK	SYNCHRONIZE
	BYTE-LENGTH	UNITS

Table 2: Core-Set Attribute List Examples

Entity Type	Core-Set Attribute List	
IMAGE-DATASET	SCOPE-OWNER	SENSITIVITY
	SCOPE-CHANGED	ID-NUMBER
	DATE-CREATED	IMAGE-DATASET-TYPE
	DATE-CHANGED	

Creating and Accessing Entities

In System Dictionary, you can create, delete, modify, and retrieve entities.

To create an entity, the dictionary must be open in *shared* or *exclusive update* mode. When you create an entity, you must specify its external name, its entity type, an attribute list, and values for each attribute in the list. The attribute list may contain any attribute associated with the entity type, plus any alias attribute. Variable-length attribute values are assigned later, after you create the entity. When you create an entity, you can specify an internal name. If you do not specify an internal name, it will default to the external name.

Modifying an entity means changing the entity's external name or assigning new attribute values. To modify an entity, the dictionary must be open in *shared* or *exclusive update* mode.

To delete an entity, the dictionary must be open in *shared* or *exclusive update* mode. When you delete an entity, relationships, variable-length attribute values, aliases, and other information are also deleted. To retrieve an entity, the dictionary must be open in *read*, *read only*, *shared update*, or *exclusive update* mode. One reason to retrieve an entity is to determine whether it exists. Another reason is to retrieve one or more of the entity's attribute values. You retrieve the attribute values by specifying the list of attributes for which values are to be returned.

When the dictionary is open in *read*, *read only*, *shared update*, or *exclusive update* mode, you can retrieve information about dictionary structure that is needed to create, modify, delete, or retrieve entities. When you create or access an entity, you must specify the entity type. System Dictionary allows you to retrieve the list of entity types available in the dictionary. You can also retrieve information about the attributes associated with an entity type so that you can assign values to the appropriate attributes. Since you can assign any alias or variable-length attribute to any entity, System Dictionary also allows you to retrieve information about these attributes.

Detailed instructions for creating, deleting, modifying and retrieving entities are located in other HP System Dictionary/XL manuals. Refer to the *HP System Dictionary/XL SDMAIN Reference Manual* for instructions to do these operations with the System Dictionary user interface, or to the *HP System Dictionary/XL Intrinsics Reference Manual*, if using the System Dictionary intrinsics.

Relationships

Relationships are dictionary definitions that express a logical connection between the real-world objects of an information network. Most relationships involve two entities and are called binary relationships. System Dictionary supports binary relationships, as well as N-ary relationships that involve from three to six entities. A relationship serves two main purposes:

- Relationships express a connection between entities. The fact can be expressed that a VPLUS form

named WINERY contains a field named VINTAGE by creating the relationship WINERY contains VINTAGE using the relationship type FORM contains ELEMENT. This dictionary definition documents the true connection between the form and the field.

- Relationships describe a context in which an entity is used. As already mentioned, an entity has attribute values that provide a general description of an object. Relationships also have attribute values. They document the entity in a specific context or usage.

A relationship definition has three parts:

- An ordered list of the entities involved in the relationship.
- A relationship type, a dictionary category to which the relationship belongs. A relationship type is an ordered list of entity types paired with a connecting name called a relationship class.
- Attribute values that further define and characterize the relationship (or, in some cases, one of the entities in the relationship).

These relationship components are discussed next.

Specifying Entity Lists

A relationship does not have a name. It has a list of names representing the entities involved in the relationship. Before you can create a relationship, all the entities in the entity list must already exist. Also, if you delete one of the entities involved in a relationship, System Dictionary automatically deletes the relationship. The order of the entity list is significant. The relationship CUSTOMER-ADDRESS contains LAST-NAME, for example, is not the same as LAST-NAME contains CUSTOMER-ADDRESS.

Specifying Relationship Types

Relationships belong to categories called relationship types. A relationship type has two parts: an ordered list of entity types that serves as a template for defining the entity list, and a relationship class that expresses the action or connection of the relationship. When you create or access a relationship, you must specify the relationship type.

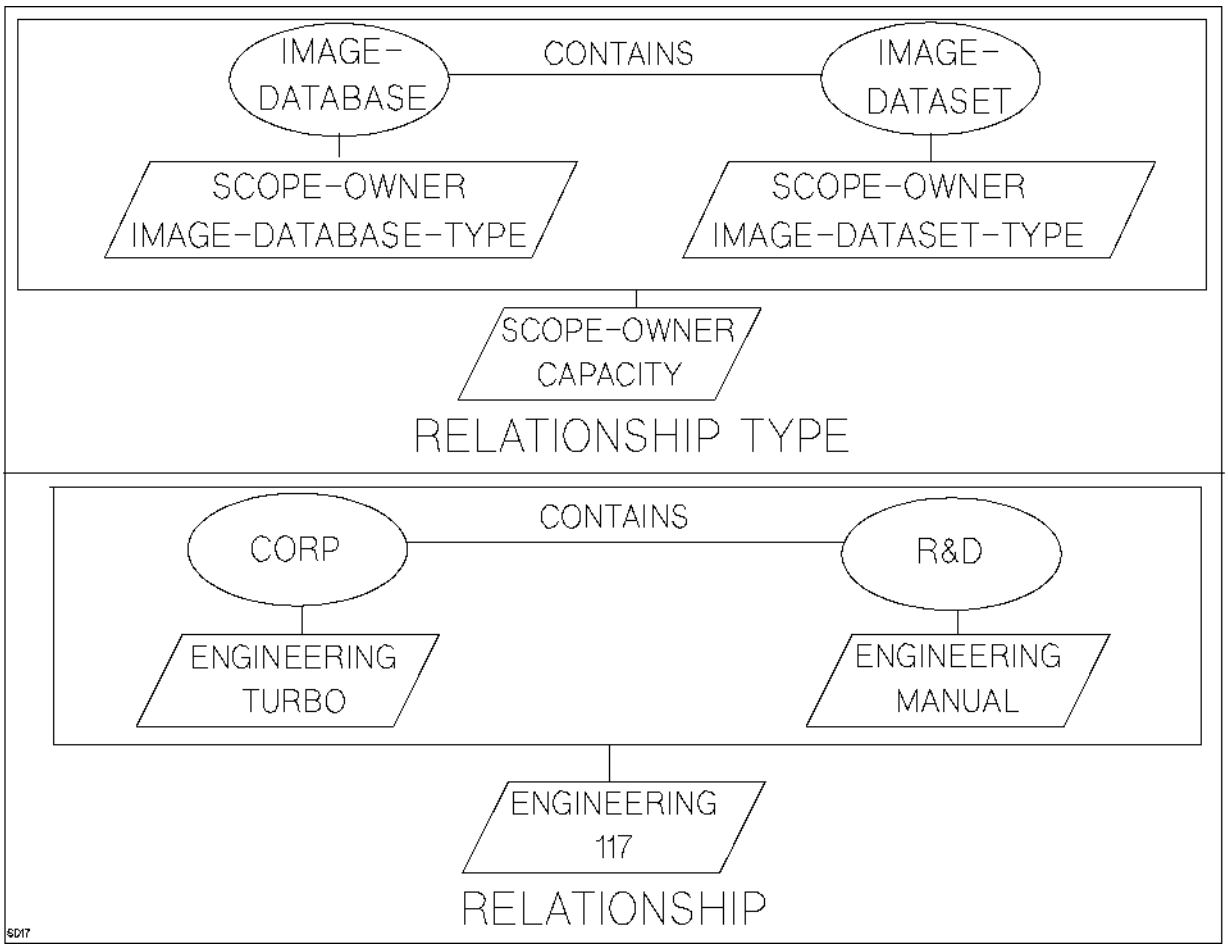
For example, RECORD contains ELEMENT is a relationship type in the System Dictionary core set. This relationship type requires that all relationships of this type must involve two entities. The first entity must be of type RECORD, and the second must be of type ELEMENT. The relationship class (contains) expresses the fact that the record contains the element.

When you create or access a relationship, you must use a relationship type that already exists in the dictionary or System Dictionary issues an error. System Dictionary comes with a built-in set of relationship types called core set relationship types. You can also add an extended set of relationship types. You can create relationships using any relationship type in the core set or extended set. Figure 3-4 illustrates an example of a relationship type and a relationship occurrence of that relationship type. In the example, IMAGE-DATABASE and IMAGE-DATASET are the entity types of the relationship type, CONTAINS is the relationship class, and SCOPE-OWNER, IMAGE-DATABASE-TYPE, and IMAGE-DATASET-TYPE are attributes of the entity types. SCOPE-OWNER and CAPACITY are attributes of the relationship type.

In the RELATIONSHIP shown, ENGINEERING, TURBO, and MANUAL are the attribute values of the entities CORP and R&D, while ENGINEERING and 117 are the attribute values of the relationship.

Click here to view figure.

Figure 3-4. Relationship Types vs. Relationships



Specifying Relationship Classes

In System Dictionary, a relationship type and, consequently, all relationships belonging to the type are qualified by a relationship class that describes the connection or action of the relationship. The System Dictionary core set includes relationship types like contains, uses, redefines, and other commonly used connecting words.

The relationship class is required when more than one relationship type in the dictionary involves the same ordered list of entity types.

In the core set, for example, there are three relationship types involving element pairs:

ELEMENT contains ELEMENT

ELEMENT redefines ELEMENT

ELEMENT references ELEMENT

The first type describes a parent-child relationship. The second describes elements that share common storage space in a program. The third type documents an element that references another element, as in a Pascal type reference. When creating or retrieving relationships defined with non-unique entity type lists like these, you must specify the relationship class as a qualifier or System Dictionary issues an error.

Remember that a relationship type is an entity-type list paired with a relationship class. Many of these pairs are built into the dictionary core set, and you can extend the dictionary structure by adding your own pairs. When you create a relationship, however, you cannot mix and match entity-type lists and

relationship classes. You must use an existing pair, or System Dictionary issues an error stating that the relationship type does not exist.

Relationship Attribute Values

Relationships have attribute values that describe the relationship definition itself, such as when it was created, who "owns" it, and so on. You can also use the attribute values of a relationship to describe one of the entities involved in the relationship. Attribute values can provide a description of the entity in a particular context.

Table 3-3 shows some sample relationships of the type FORM contains ELEMENT. Each "row" in the table represents a relationship involving two entities, shown in the first two columns. The other columns contain values for the attributes SCOPE-OWNER and FIELD-ENHANCEMENT. The SCOPE-OWNER attribute applies to the relationship definition itself (who owns it in the dictionary). The FIELD-ENHANCEMENT value refers to an object.

Table 3: Relationships of type FORM contains ELEMENT

FORM	ELEMENT	SCOPE-OWNER	FIELD-ENHANCEMENT
CUSTOMERS	LAST_NAME	DA	N
SALES	PURCHASE_PRICE	RON	HI
BURGERS	PURCHASE_PRICE	KELLY	HIU
CHEESE_CODES	NAME	SAM	B

N-Ary Relationships

System Dictionary allows relationships that involve up to six entities. Relationships involving more than two entities are called N-ary relationships.

An example of an N-ary relationship type is the core set type IMAGE-DATASET ELEMENT ELEMENT IMAGE-DATASET IMAGE-DATABASE chains. This relationship type defines an IMAGE data base chain by including the detail data set, search item, sort item, master set, and data base in a single relationship.

An N-ary relationship allows you to define multiple relationships in a single definition. It also prevents part of the definition from being left out. You cannot create an N-ary relationship unless you specify all the entities. If it is necessary to omit one of the entities in an N-ary relationship, you may use a slash (/) to indicate a null entity.

Creating and Accessing Relationships

You can create, delete, modify, and retrieve relationships. To create, delete, or modify a relationship, the dictionary must be open in shared or exclusive update mode. To retrieve a relationship, the read and read only mode are sufficient.

When you create a relationship, you must specify a list of already existing entities that will be involved in the relationship. You must specify the relationship type as an ordered list of entity types corresponding to the entity list. If the entity type list you specify is not unique in the dictionary (for example, ELEMENT ELEMENT), you must qualify the entity type list by relationship class (for example, contains). When you create a relationship, you can specify a list of attributes and their values. The list may contain any attribute associated with the entity type, plus any alias attribute. Variable-length attribute values are to be assigned later, after you create the relationship.

To retrieve a relationship, the dictionary can be open in read, read only, shared update, or exclusive update mode. You retrieve a relationship to determine whether it exists, or to retrieve one or more of its attribute values. You can request a value for any attribute associated with the relationship type, or for any alias or variable-length attribute.

When the dictionary is open in read, read only, shared update, or exclusive update mode, System Dictionary allows you to retrieve dictionary structure information that is needed to create, delete, modify, and retrieve relationships. You can retrieve a list of relationship types and relationship classes available in the dictionary. Also, you can retrieve the list of attributes associated with a relationship type, and a list of the alias and variable-length attributes since you can assign these to any relationship.

Detailed instructions for creating, deleting, modifying, and retrieving relationships are located in other HP System Dictionary/XL manuals. Refer to the HP System Dictionary/XL SDMAIN Reference Manual (32256-90001) for instructions to do these operations with the System Dictionary user interface, or to the HP System Dictionary/XL Intrinsics Reference Manual (32256-90002), if using the System Dictionary intrinsics.

Entity Types

An entity type is a template for defining entities in the dictionary. System Dictionary comes with a set of built-in entity types, called core set entity types. The entity types of the core set are intended to be comprehensive enough to define most subsystem objects. You can also add new entity types if desired. In System Dictionary, the ability to add new entity types is restricted to users with a special capability (the extend scope-right).

An entity type has two parts:

- The entity type name.
- A list of attributes associated with the entity type.

You can think of entity types as categories. All IMAGE data base entities, for example, are grouped under the entity type IMAGE-DATABASE. You can also view an entity type as a qualifier. If a VPLUS form and an IMAGE data set both happen to be named CUSTOMERS, you can define the form as an entity of type FORM, and the data set as an entity of type IMAGE-DATASET. Since it is possible to have multiple entities with the same name, you must qualify an entity by specifying its entity type.

You cannot delete one of the entity types involved with the relationship type unless you explicitly request that all the relationship types involving that entity type are to be deleted.

Specifying Attribute Lists

Entity types have attribute lists such that each entity of the type must have a value for each entity in the list. Every entity type has built-in attributes, called special attributes (special attributes are discussed later in this chapter). When you create an entity type, System Dictionary automatically associates these attributes with the new entity type.

In addition, each entity type may have other attributes associated with it. For each attribute in an entity type's attribute list, there must be a corresponding attribute value assigned to each entity of that type. You can add attributes to an entity type's attribute list by creating type-attribute associations (discussed later).

Creating and Accessing Entity Types

You can create, delete, modify, and retrieve entity types. The actual instructions for doing these operations are located in other HP System Dictionary/XL manuals. Refer to the HP System Dictionary/XL SDMAIN

Reference Manual (32256-90001) for instructions to do these operations with the System Dictionary user interface, or to the HP System Dictionary/XL Intrinsics Reference Manual (32256-90002), if using the System Dictionary intrinsics.

Relationship Types

A relationship type is a template for defining relationships in the dictionary. A relationship type has three parts:

- An ordered entity type list.
- A relationship class.
- An attribute list.

Like an entity type, you can think of a relationship type as a category or grouping. Relationships between IMAGE data bases and data sets are grouped under the relationship type IMAGE-DATABASE contains IMAGE-DATASET. You can also view a relationship type as a qualifier. The entity type list of the relationship type qualifies the entities in the relationship. You can have multiple relationships between entities named ORDERS and INVENTORY provided you qualify them by different relationship types.

Specifying Entity Type Lists

A relationship type does not have a name. It has a list of names representing the entity types involved in the relationship type. Before you can create a relationship type, all the entity types in the entity type list must already exist. Also, you cannot delete one of the entity types involved with the relationship type unless you explicitly request that all the relationship types involving that entity type are to be deleted.

Specifying Relationship Classes

In System Dictionary, a relationship type is qualified by relationship class. The relationship class describes the connection or action of a relationship. In the relationship type IMAGE-DATABASE contains IMAGE-DATASET, for example, the relationship class is contains. It expresses the fact that, in each relationship of this type, the first entity contains the second.

To create a relationship type, you must specify a relationship class that already exists in the dictionary. You cannot delete a relationship class without explicitly requesting the deletion of all the relationship types of that class.

Specifying Attribute Lists

Relationship types have attribute lists such that each relationship of the type must have a value for each attribute in the list. Every relationship type has six built-in attributes, called special attributes (special attributes are discussed later in this chapter). When you create a relationship type, System Dictionary automatically associates these attributes with the new relationship type.

In addition, each relationship type may have other attributes associated with it. For each attribute in a relationship type's attribute list, there must be a corresponding attribute value assigned to each relationship of that type. You can add attributes to a relationship type's attribute list by creating type-attribute associations (discussed later in this chapter).

Creating and Accessing Relationship Types

You can create, delete, modify, and retrieve relationship types. Detailed instructions for doing these

operations are located in other HP System Dictionary/XL manuals. Refer to the HP System Dictionary/XL SDMAIN Reference Manual for instructions to do these operations with the System Dictionary user interface, or to the HP System Dictionary/XL Intrinsics Reference Manual, if using the System Dictionary intrinsics.

Relationship Classes

A relationship class is a name in the dictionary structure that you can use to express the action or connection of a relationship type. See the Syntax discussion in Chapter 2 for more information on relationship class names. The System Dictionary core set contains several commonly used relationship classes. You can add relationship classes to the dictionary, and you can change external names of the core set classes.

In System Dictionary, relationship types belong to a relationship class. For example, it is possible to retrieve all the relationship types that belong to a relationship class. To get a list of all the relationship types in the dictionary, you must first get the list of relationship classes and retrieve the relationship types of each class.

You can create, delete,modify, and retrieve a relationship class. The actual instructions for doing these operations are located in other HP System Dictionary/XL manuals. Refer to the HP System Dictionary/XL SDMAIN Reference Manual for instructions to do these operations with the System Dictionary user interface, or to the HP System Dictionary/XL Intrinsics Reference Manual, if using the System Dictionary intrinsics.

You cannot delete a relationship class without explicitly requesting the deletion of all the relationship types of that class.

Attributes

An attribute is a characteristic, property, or description of an entity or relationship. In System Dictionary, attributes are dictionary structures that allow attribute values to be assigned to entities and relationships. System Dictionary has a core set of attributes that are built into every dictionary. You can also create an extended set of attributes if desired.

An attribute definition has four parts:

- The attribute name.
- The attribute type (alias, Boolean, character, floating, integer, or variable).
- The attribute length--a value, dependent on the attribute type, that specifies the maximum length allowed for values of the attribute.
- An optional list of edits that specify the default and allowable values for attributes of type Boolean, character, floating, and integer.

You can associate attributes of type Boolean, character, floating, or integer with an entity type or relationship type. Attributes of type alias and variable are "free-floating" in that they are never associated with an entity type or relationship type but you can assign them to any entity or relationship. In either case, attributes serve as templates for defining the attribute values of entities and relationships. Attribute values do not exist independently. They are always associated with a specific entity or relationship. When you delete an entity-type-attribute association, System Dictionary must remove the corresponding attribute value from each entity of the affected type.

Since attributes are part of the dictionary structure and are available to you, you should avoid obscure or

redundant attribute names. This is especially true for alias and variable attributes, because you can assign them to any entity or relationship.

Specifying Attribute Type and Length

System Dictionary supports six attribute types: alias, Boolean, character, floating, integer, and variable.

Boolean attributes are used to store true-or-false values. Internally, Boolean values are stored in a single byte, but their values are typically displayed as TRUE or FALSE. An example of a Boolean attribute in the core set is the BLANK attribute. BLANK is an attribute of all entities of type ELEMENT, and it specifies whether zero values should be displayed as blanks or zeros.

For example, suppose a COBOL program contains a variable named YEAR-TERMINATED that displays the year in which an employee leaves the company. If an individual is currently employed, the value of this field is zero but is displayed as blanks. To document this variable in the dictionary, you would need to name an entity named YEAR-TERMINATED of type ELEMENT. The attribute list of the ELEMENT entity type (shown in Table 3-2) includes the BLANK attribute. For the YEAR-TERMINATED entity, you would need to set this attribute to TRUE.

Integer and floating attributes store numeric values. Integer attributes can have a length of 2 bytes (16 bits) or 4 bytes (32 bits). A floating attribute can have a length of 4 bytes or 8 bytes.

Attributes of type character store fixed-length alphanumeric values. A character attribute can have a length between 1 and 255 bytes inclusive.

Specifying Attribute Edits

An attribute of type Boolean, character, floating, or integer can have a list of edits that specify the default and allowable values of the attribute. The first value in the edit list indicates the default value. Other values indicate allowable values. To specify a default value while allowing any other value, you must supply an edit list with only one edit (the default value).

Variable-Length Attributes

A variable-length attribute is an attribute of type character whose length is unrestricted. Variable-length attributes are "free-floating" in that they never appear in the attribute list of an entity type or relationship type but you can assign them to any entity or relationship.

Variable-length attributes typically contain descriptions, defaults, edit masks, picture clauses, or other free-format values. Since not every entity or relationship needs these values, and since the values can be quite large, System Dictionary reserves storage space for variable-length attribute values only if you explicitly assign them to an entity or relationship. When you delete a variable-length attribute value, System Dictionary releases the storage space.

DESCRIPTION, for example, is a variable-length attribute in the System Dictionary core set. Table 3-4 shows some sample entities and their DESCRIPTION attribute values. In the first and second columns are the entity and entity type names. The third column contains the description. As the table suggests, if a DESCRIPTION attribute value exists, it is linked to a specific entity of a specific type. Unlike the sample attributes in Table 3-1 and Table 3-3, the DESCRIPTION attribute is not associated with any entity type

or relationship type.

Table 4: DESCRIPTION Attribute Values

Entity	Entity Type	DESCRIPTION
CUSTOMERS	FORM	Customer info for Admin group
ORDERS	IMAGE-DATABASE	Order-tracking data base
LAST_NAME	ELEMENT	Last name is upshifted
DATE	RECORD	Divided into month/day/year elements

You can assign a DESCRIPTION attribute to the entity LAST-NAME of type ELEMENT. It is up to you whether other entities of type ELEMENT also have DESCRIPTION attribute values.

When you request a variable-length attribute value of an entity or relationship you specify, two outcomes are possible. The first outcome is that the entity or relationship you specify has a value for the attribute you specify (DESCRIPTION, for instance), and the value is returned. The second possibility is that the value you requested was never assigned to the entity or relationship you specified, and this is reported as an error.

Alias Attributes

An alias is an alternate entity name that reflects the entity's usage in a particular subsystem. In System Dictionary, an alias is a fixed-length, 32-character attribute that you can assign to any entity or relationship. Alias attributes never appear in the attribute list of an entity type or relationship type. Since not every entity or relationship needs aliases, storage space for alias attributes is assigned as needed.

Table 3-5 shows a list of entities that have IMAGE-ALIAS attribute values. IMAGE-ALIAS, for example, is an alias attribute that you can use to document an alternate name used in an IMAGE data base. In the first column are the entity names, with their entity types in the second column. The entity CUSTOMER-LAST-NAME of type ELEMENT has an IMAGE-ALIAS value of CUST-LAST-NAME. This alternate name was chosen because IMAGE item names must be 16 characters or less. For the entity ORDERS-MIS-DATABASE, the IMAGE alias is ORDERS, because IMAGE data base names must be 6 characters or less. The IMAGE alias of the BURGER_NAME entity is BURGER-NAME, because IMAGE does not allow underscores in item names.

Table 5: IMAGE-ALIAS Attribute Values

Entity	Entity Type	IMAGE-ALIAS
CUSTOMER-LAST-NAME	ELEMENT	CUST-LAST-NAME
ORDERS-MIS-DATABASE	IMAGE-DATABASE	ORDERS
BURGER_NAME	ELEMENT	BURGER-NAME

When you retrieve an alias attribute value of an entity or relationship, a value is always returned. If the alias exists, a 32-byte value containing the alias name is returned. If the alias does not exist, a blank value is returned. You can delete an alias attribute value by replacing it with all blanks. When you delete an alias attribute value, its storage space is released.

Special Attributes

Every entity type and relationship type has a set of special attributes automatically associated with it as part of the process by which it is created. The set of attributes is not the same for entity types and relationship types. Most of these attributes have values automatically inserted when you create an occurrence. A typical example of this is the attribute date-created. This attribute is automatically associated with every entity type and relationship type as it is created, and whenever an entity or a relationship is created, the current date is assigned as the value for that attribute. The following attributes are designated special attributes:

Special Attributes for Entity Types	Special Attributes for Relationship Types
scope-owner	scope-owner
date-created	date-created
date-changed	date-changed
scope-changed	scope-changed
sensitivity	sensitivity
id-number	relationship-position

Some characteristics that make these attributes "special" are:

- * They are built into every core set entity type and relationship type.
- If you extend the dictionary by creating new entity types and relationship types, System Dictionary automatically assigns these attributes to them, and they cannot be removed later.
- You cannot add the id-number attribute to a relationship type, and you cannot add the relationship-position attribute to an entity type.
- When you create or modify an entity or relationship, you cannot assign values to scope-changed, date-created, and date-changed, because these attributes are managed exclusively by System Dictionary. Also, you cannot assign a value to the scope-owner attribute when you create an entity or relationship; System Dictionary sets this attribute. You can, however, assign an existing entity or relationship to a new scope-owner by modifying its scope-owner attribute.

Type-Attribute Associations

You can assign an attribute of type Boolean, character, floating, or integer to any entity type or relationship type. This is referred to as creating a type-attribute association. When an attribute is associated with an entity type or relationship type, all entities or relationships of that type must have a value for that attribute. Type-attribute associations build the attribute lists of entity types and relationship types.

System Dictionary has a built-in set of type-attribute associations. These type-attribute associations are part of the core set and cannot be deleted. You can extend the dictionary by creating other type-attribute associations. You can add attributes to core set entity types and relationship types or to new entity types and relationship types. You can later remove these associations from the extended set.

Entity-Type-Attribute Associations

Entity types are templates for defining the set of attribute values in each entity of that type. When you create a type-attribute association, the template is changed and a corresponding change must be made in the attribute values of each entity of that type. The change, which involves assigning the attribute's

default value to each entity, happens during restructuring when the dictionary is closed. (Restructuring of the dictionary is discussed later in this chapter.)

Similarly, when you delete an entity-type-attribute association, System Dictionary must remove the corresponding attribute value from each entity of the affected type.

Relationship-Type-Attribute Associations

Relationship types are templates for defining the attribute values of relationships. When you create or delete a relationship-type-attribute association, the template is changed, and a corresponding change must be made in each relationship of the affected type.

Naming Mechanisms

System Dictionary provides a variety of naming conventions that support the definition of the following:

- Modifiable and non-modifiable names
- Alternate entity names called synonyms
- Subsystem-related names called aliases
- Internal numbers that programmers can use

A discussion of these naming mechanisms follows.

Internal and External Names

Every dictionary definition or structure that you can name has two names, an internal name and an external name. The internal name is a stable, non-modifiable name, while the external name can be changed. You can also modify the external names of core set structures. This allows the dictionary to be customized to meet the requirements of a particular application, or localized to your native language, while still retaining a compatible set of internal names.

When the dictionary is open, you specify a name mode, either internal or external. If you select internal name mode, all definitions and structures are accessed by their internal names only. In external name mode, you can see only external names.

Primary Names

In System Dictionary, a primary name is the name of an entity as opposed to its synonyms or aliases. Since an entity has two names--internal and external--there are actually two primary names for each entity. In internal name mode, an entity's primary name is its internal name. In external name mode, an entity's primary name is its external name. The primary name of the entity is the name returned when System Dictionary returns an entity name.

Synonyms

A synonym is an alternate name that uniquely identifies an entity. An entity may have multiple synonyms, but a synonym can refer to only one entity. You can think of a synonym as a "pointer" that points to an entity's primary name. All synonyms of a given entity, point to the same primary name. You can use synonyms in any context where entity names are allowed. You can modify, delete, or retrieve an existing entity by using the synonym in place of the entity name.

A synonym has three parts:

- The synonym name.
- The name of the entity that the synonym points to.
- The entity type of the referenced entity. A synonym implicitly has the same entity type as the entity it references.

A synonym name must be 32 characters or less. It may not contain embedded blanks, and you cannot use the restricted characters in Table 2-1. Because System Dictionary allows you to use entity names and synonyms interchangeably, synonyms are subject to the same uniqueness requirements as entities. Two synonyms of a given entity type cannot have the same name, and a synonym may not have the same name as an entity of the same type.

A synonym points to a specific entity of a specific type. To create a synonym, the underlying entity must already exist.

Aliases

An alias is an alternate name related to an entity's usage in a particular subsystem. In System Dictionary, aliases are stored in attributes of type alias. Alias attribute values are 32-byte character strings that you can assign to any entity or relationship.

Unlike other dictionary names, an alias may contain embedded blanks. It may also contain any combination of uppercase and lowercase characters. System Dictionary always upshifts entity names, but aliases allow a case-sensitive name to be kept exactly as it appears in a program.

An alias may contain any special characters, even the restricted characters in Table 2-1. There are no uniqueness requirements for aliases. An alias name can be the same as its entity name if desired, and any number of entities and relationships can have the same alias.

When an alias is assigned to an entity, it documents an alternate name that should meet the naming restrictions of the target subsystem (although System Dictionary does not check the name against the subsystem restrictions).

When you assign an alias to a relationship, it provides an alternate name for the second entity in the relationship. For example, consider the relationship CUSTOMERS contains LAST-NAME of type FORM contains ELEMENT. You can give the relationship a VPLUS-ALIAS attribute that contains the value LAST_NAME. In this case, the relationship-level alias documents an alternate name for the LAST-NAME entity.

Internal Numbers

Every definition and structure in the dictionary has an internal number that System Dictionary uses to optimize storage and retrieval. You can retrieve these numbers by dictionary intrinsics and use them in place of names for improved performance.

Relationship Identification

A relationship does not have a name. Instead, it is identified by a list of entity names. A relationship type is similarly identified by a list of entity type names rather than by a single name. Relationship classes

further qualify relationships and relationship types.

Each relationship in the dictionary has a single internal number that you can use in place of the relationship's entity list. Similarly, each relationship type has a single internal number that you can use in place of the entity type list. A single internal number therefore, replaces up to six names in a relationship or relationship type.

Restructuring the Dictionary

Whenever you modify the dictionary structure, that is, whenever you add, delete, or modify any entity types, relationship types, relationship classes, attributes, entity type/attribute pairs, or relationship type/attribute pairs, System Dictionary requires that you use a process called restructuring to incorporate those changes into the working dictionary. Restructuring is essentially a compilation of the dictionary structure into a form easily accessible by the dictionary intrinsics, and reformatting any dictionary occurrences that are affected by the structure changes.

NOTE The entire dictionary structure is not necessarily recompiled. Only those definitions that are affected by the changes made to the dictionary structure will be involved in the restructuring process.

Many structure changes may be incorporated into a single restructuring, thereby increasing the overall efficiency of the restructuring operation. This grouping of structure changes is controlled by allowing structure changes to be made only while the dictionary is open in the exclusive customization mode.

You may make as many structure changes as desired during a session open in this mode. However, the only operations allowed in this mode are:

- Structure changes
- Retrieval of structure information
- Limited retrieval of security information

You cannot perform any operations involving dictionary occurrences while the dictionary is open in exclusive customization mode. When you close the dictionary, any restructuring required is performed prior to the actual termination of access. Opening the dictionary with this special mode implies exclusive access to the dictionary and is limited to the DA scope or scopes with extend capability.

A dictionary structure change can affect occurrences in the dictionary in several ways. For example, deleting an entity type causes any existing occurrences of that type to be deleted and adding an attribute to an existing entity type causes all occurrences of that type to be modified. This restructuring of the dictionary and reformatting of occurrences is completely transparent to you. The different types of changes are handled in the following manner:

- Adding an attribute to an entity or relationship type causes all occurrences of that type to receive a defaulted attribute value for the new attribute.
- Deleting an alias or variable length attribute causes all attribute values associated with the attribute to be deleted.
- Deleting an attribute from an entity or relationship type causes that attribute to be removed from the attribute list of the entity or relationship type and causes the attribute value of the attribute to be removed from all occurrences of that type.
- Changing the size of an attribute causes the attribute value to be adjusted in size for all occurrences of entity and relationship types that have that attribute.
- Deleting a relationship class causes any relationship types that use that class to be deleted.
- Deleting an entity type causes any relationship types that include that entity type to be deleted.

- Deleting an entity type or relationship type causes all occurrences of that type to be deleted.

Dictionary Size Limits

The number of System Dictionary components (core set and extended set combined) that may exist in the dictionary at any one time is subject to certain limits. The following list summarizes the limits on dictionary components.

Domains	128
Versions in a Domain	128
Scopes	128
Entity Types	256
Relationship Classes	128
Relationship Types	512
Attributes	1024
Alias Attributes (subset of Attributes)	128
Entity Type - Attribute Associations per Type	128
Relationship Type - Attribute Associations per Type	128
Entity Type - Attribute Associations per Dictionary	5120
Relationship Type - Attribute Associations per Dictionary	5120
Entity Occurrences per Dictionary	1000000
Relationship Occurrences per Dictionary	1400000

4 Domains and Versions

Overview

This chapter provides a description of domains and versions, which are the spaces within System Dictionary where the entities and relationships reside. Information on security for domains and versions is discussed in Chapter 5 of this manual.

Domains and Versions

One important function of a data dictionary is to maintain standardization of the data in an information system. As shown in the previous chapter, System Dictionary includes this capability by providing you with a standard set of structure definitions, but also the means to extend them to meet your particular needs.

Similarly, System Dictionary provides you with the means for maintaining a standard set of occurrences--the entities and relationships. However, though most of the occurrences in a data dictionary are meant to be shared by all users, there are sometimes circumstances, applications, or users that need separate sets of occurrences.

System Dictionary handles these needs by allowing multiple spaces called **domains** within a single dictionary. Domains are similar in some ways to accounts in the MPE file system, and you can use them as separate partitions for different applications, or as "name spaces" to separate duplicate names used for different purposes.

Figure 4-1 shows examples of domains that could exist within a particular configuration of System Dictionary. The center area represents the **common domain**, which is the domain provided with System Dictionary. The other spaces represent user-created domains, called **local domains**. The common domain and local domains are discussed further on in this chapter.

Figure 4-1. Dictionary Domains

Using Domains

You can use domains to satisfy a number of your needs. Some of the more common uses for domains are:

- Providing partitions for separate applications, or "name spaces" for different definitions which have the same name.
- Providing spaces for temporary definitions which may be deleted all at once by simply deleting the domain that contains them. This is similar to storing files in a temporary account in MPE and then deleting the account. Note that you can also do this at the version level, just as you can create a temporary group within an account and then purge when you no longer need it. Versions are discussed further on in this chapter.

Using Domains as Name Spaces

You can use domains as "name spaces" to avoid naming conflicts that can occur when two or more of you want to use the same name for different purposes. For example, the Personnel and Manufacturing

departments of a company both want to use an entity called P-NUM. The Personnel department defines P-NUM as a personnel number, while the manufacturing department wants to use it for part number. System Dictionary easily handles this conflict by allowing you to create a domain for each application.

Within a single domain, System Dictionary does not allow two occurrences of the same type to have the same name. For example, you cannot have two entities called PRODUCT-NAME of the same entity type (for example ELEMENT) in the same domain. The same is true for relationships of the same relationship type. However, System Dictionary does allow two entities that use the same name if the entities are of different entity-types in the same domain. For example PRODUCT-NAME of type RECORD, and PRODUCT-NAME of type ELEMENT do not conflict with each other. This also applies to relationships.

Using Domains as Partitions

Just as you can use an account in the MPE file system to contain a set of related files, you can use a domain to contain a set of occurrences a specific application uses. For example, you may want to have similar sets of occurrences for the Marketing, Personnel, Manufacturing, and R&D departments. You can easily accomplish this by creating a domain for each of them, as shown in figure 4-1. Note that this also keeps the occurrences for that application separate from those of all others in the dictionary.

The Common Domain

System Dictionary includes a built-in domain called the common domain, which is created when the dictionary is initialized. You can use this domain as the standard dictionary, that is, the space where all the standard entities and relationships reside, and it is accessible by any dictionary user.

The common domain in each dictionary may be its only domain, or instead, serve as the domain containing information common to any number of local domains. (See the discussion on "linking", further on in this chapter.) If you have no need for partitioning the dictionary, you would use the common domain only. You can ignore the concept of domains, and when opening a dictionary, allow System Dictionary to default to the common domain.

Local Domains

You can create your own domains, called local domains, which you can use as partitions or name spaces, as described above. Local domains can be completely self-contained, that is, the occurrences contained in them can have their own attribute values. Instead of the occurrences in each domain having their own unique set of attribute values, which takes up more space and can make a dictionary "non-standard", System Dictionary allows occurrences in local domains to share attribute values with occurrences in the common domain. This saves space and creates a more standard dictionary, which is also easier to maintain. Sharing attribute values is discussed further on in this chapter.

Creating Domains

You can create domains through either the System Dictionary intrinsics or an SDMAIN command. No matter what method you use, an empty domain is created and given a user-specified name.

Domains and the Dictionary Structure

Although you can use the domain feature of System Dictionary to create separate sets of entities and relationships, you cannot use it to create separate dictionary structure definitions. All domains in a dictionary exist in the same dictionary structure, and therefore, regarding the structure, entities and relationships are always specified and used the same way, no matter which domain you are using.
Accessing Domains

A domain is part of the "access path" to System Dictionary. Therefore, you must specify a domain each time System Dictionary is open in a mode that allows access to domains. Unlike domains that you name, the common domain does not have a name (its name is blank), and you access it by using blanks when specifying the domain to use.

Because only one dictionary can be open at a time through SDMAIN, you are restricted to using only one domain at a time. You can switch to another domain (modify the current access path), but the rule that you can use only one domain at a time is enforced. Although you can establish multiple access paths to one or more dictionaries simultaneously via the System Dictionary intrinsics, and can switch between paths, you can only specify one domain at a time per access path, and only one of these paths may be used at a time. Therefore, the rule that you may use only one domain at a time is still enforced. In any case, you must work entirely within the specified domain except when using a local domain that has links to the common domain. (See the discussion on "linking", further on in this chapter.) Note that you are able to access only the occurrences in the domain specified in the current access path.

Versions

Some applications require multiple versions of dictionary occurrences. System Dictionary provides a version feature that allows multiple versions of the same entities and relationships to exist simultaneously within a single domain. A version in the dictionary is a name space within a domain, and you can think of it of as a partition of the domain, or as a copy of the entities and relationships in the domain.

Versions as Partitions

Versions are named partitions of a domain, and you can identify them by creating names for them. These names must be valid System Dictionary names, and can take the form of a number such as '051743', or a name such as 'TEST_VERSION_ONE', etc. For example, a domain named MFG could contain two versions named MFG1 and MFG2. Every domain must contain at least one version. In this sense, versions in System Dictionary are similar to groups within an MPE account. Figure 4-2 illustrates

versions within a domain. Note that this domain could be either the common domain or a local domain.

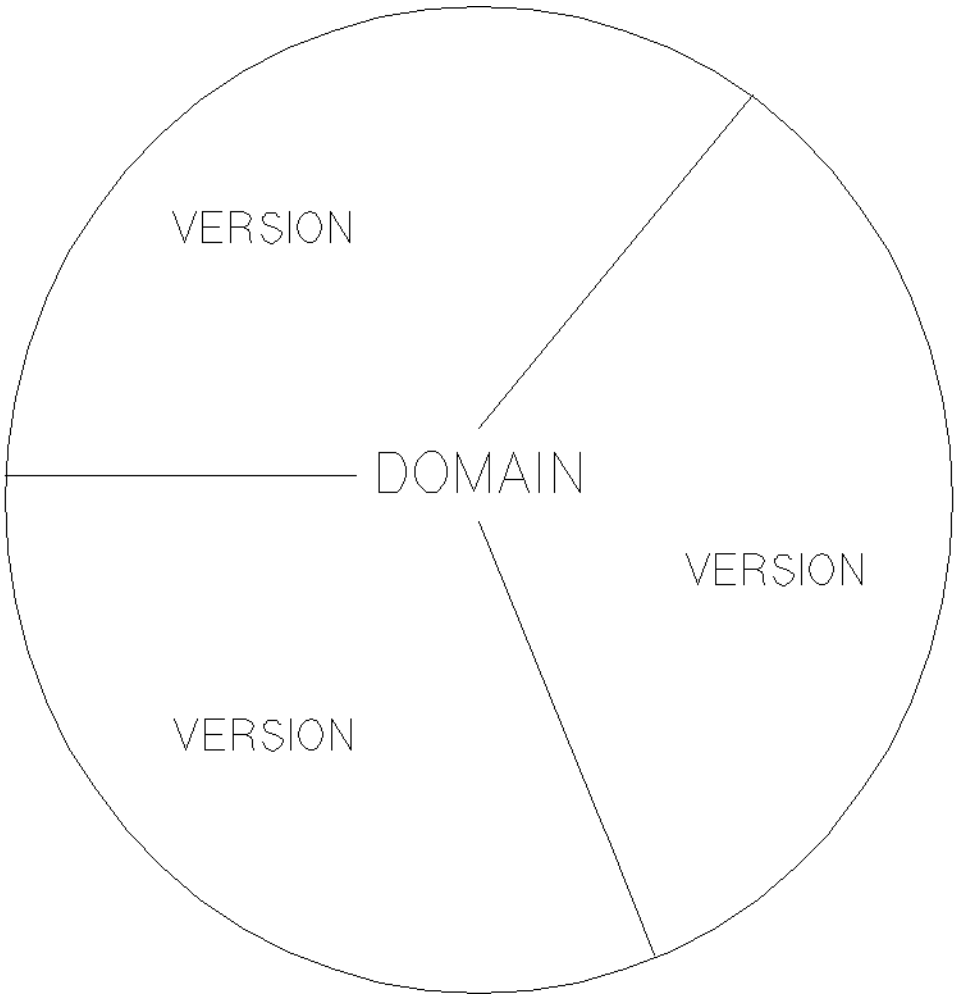


Figure 4-2. Versions

Versions As Copies

You can also think of versions as copies of the entities and relationships in a domain. In this sense, the version feature allows the creation of different versions of entities and relationships. For example, a domain called ADDRESSES contains two versions, ADRSHORT and ADRLONG. Both versions contain an entity called ZIP-CODE of type RECORD, which is described by the attribute BYTE-LENGTH. In version ADRSHORT the value of BYTE-LENGTH is 5, meaning that the entity ZIP-CODE in this version describes a 5-digit zip code. In version ADRLONG, however, the value of the attribute BYTE-LENGTH is changed to 9. The entity ZIP-CODE in version ADRLONG is therefore a different version of the entity ZIP-CODE in version ADRSHORT, and describes a 9-digit zip code.

You can copy an entire set of version occurrences from one version to another. This is useful, for example, when creating a test version with all the occurrences of the current production version. Note, however, that it is not necessary for each version of a domain to contain all the entities and relationships in that domain. A version of a domain can contain more occurrences than other versions of the domain, or fewer, or may even be empty (a newly created version is always empty).

With certain exceptions, the attribute values of a given entity or relationship may vary from one version to another. In fact, the main purpose of the version feature is to allow you to experiment with one version by changing attribute values or adding new definitions while other versions remain undisturbed.

All versions of an occurrence use the same name and occurrence-scope associations, however, because these are defined at the domain level, as are the values for the following attributes for occurrences:

- scope-owner
- sensitivity
- id-number (entities only)

The values for all other attributes are assigned at the version level, and may be different for each version of an occurrence.

Creating Versions

You can create versions through either the System Dictionary intrinsics or an SDMAIN command, and you must create them before you can add entities or relationships to them. No matter what method you use, an empty version is created and given a user-specified name.

Version Status

Each version is assigned to one of three statuses, whose characteristics are described below.

Test Status

System Dictionary automatically assigns Test status to each version when you create it, and this status is intended for use with unproven versions of resource definitions. It is the least restricted of all the statuses, and you may assign it to more than one version in a domain at the same time. You can modify dictionary occurrences in test versions, while you cannot modify those in other statuses. Therefore, you can assign Test status to any version.

Production Status

Production status is assigned by you and is intended for the version driving a production system. You can only read occurrences of this version. Only one version per domain can have Production status at any one time.

Archival Status

Archival status is also assigned by you and is intended for use with versions that have previously been production versions and need to be maintained in the dictionary for historical reference, or as insurance against current production versions that may not function properly. Archival status is similar to Test status in that you may assign any number of versions at one time. Archival status is similar to Production status in that you can only read occurrences in an archival version. Figure 4-3 shows domains that include

a possible set of versions.

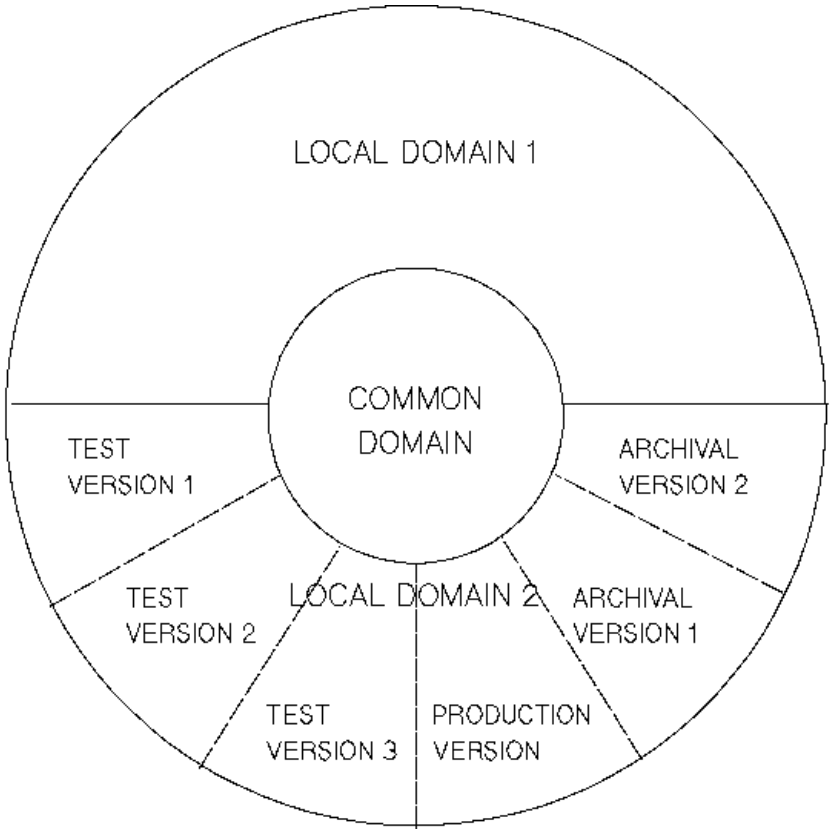


Figure 4-3. Version Examples

Changing Version Status

You may change the status of an existing version from one status to another, but you can set a version to only one status at a time. To ensure that no other user is operating in a version when you change that version's status, the dictionary must be open in Exclusive Update mode during that time.

Versions and Restructuring

All dictionary occurrences, regardless of the domain and version they are located in, use the same dictionary structure, because there is only one dictionary structure. This is an important consideration when you customize the System Dictionary structure after placing resource definitions into the dictionary. When you assign a version to either archival or production status, it is considered frozen, protecting its occurrences from modification. However, when you change the dictionary structure, all occurrences in archival and production versions are restructured like the occurrences in test versions. Adding a definition (for example, an attribute) does not cause any problems, as all existing archival information is still saved. If you delete a definition from the structure however, it is also deleted from all versions, including archival versions, thereby destroying their ability to provide accurate historical reference and backup capability. You should consider restructuring carefully, and possibly make a copy of the dictionary before restructuring.

Audit Trail

A chronological record of status changes is maintained for each version to provide an effective audit trail of system rollover. You can retrieve the following information:

- Every status change a version has undergone since its creation, and the date and time of each change.
- Every version that currently has a particular status.

Accessing Versions

Like domains, versions are part of the access path to a particular dictionary. Therefore, you must specify a version each time System Dictionary is open in a mode that allows access to versions. Note that you can specify the version by either version name or version status. As with domains, you can access only one version at a time, and you may switch versions while the dictionary is open. It is possible, through the System Dictionary intrinsics, to establish multiple access paths to the same dictionary, specifying a different version in each access, but you can specify only one version per dictionary access path. You can reference the version currently in Production status, and the versions most recently assigned to Test and Archival statuses without knowing their names, by specifying their status.

Linking Versions

Before you can link occurrences in a local domain version to occurrences in a common domain version (linking is discussed below), the local domain version must be linked to the common domain version. Once you link the versions, you can only link occurrences contained in the local domain version to occurrences in the common domain version it is linked to. For example, if you link version MFG1 of local domain OPS to common domain version A, then you can only link version MFG1 occurrences to common domain version A occurrences. Note that the linked versions are not required to have the same name.

You can move the link from a local domain version to a different version in the common domain. Before you can move the link, however, the new common domain version must contain a corresponding occurrence for each occurrence in the existing common domain version that is linked from an occurrence in the specified local domain version. For example, the local domain version MFG1 contains entities LE1 and LE2 of type RECORD, which are linked to entities CE1 and CE2 of type RECORD in common domain version A. Before you can move the links from common domain version A to common domain version B, common domain version B must contain entities CE1 and CE2 of type RECORD.

Deleting Versions

You can remove a version and the set of occurrences it contains from the dictionary as a unit. As with domains, this feature allows you to see a version as a storage space for temporary definitions, which you may delete all at once by simply deleting the version. Note, however, that you cannot delete any version having the status of Production. You would have to change the status of the version from Production to either Test or Archival to delete it. Note also, that you cannot delete a version in the common domain until you delete all links to occurrences it contains, and you delete all links to it from local domain versions.

Linking Occurrences

One of the major functions of a data dictionary is to ensure the standardization and integrity of the definitions used in an information system. However, you can use the domain feature of System Dictionary to unintentionally defeat this standardization. You can create "non-standard" dictionaries by creating identical occurrences in different domains, or by creating occurrences in different domains which, though not identical, should be. For example, an entity named LAST-NAME of type ELEMENT exists in four local domains. Each entity was created by a different user, and has a different value for one of its attributes, BYTE-LENGTH. These different values mean that there is possibly no standard length for last names described in the dictionary. If these differences are due to application requirements, the domain feature serves a useful purpose, but if they are unnecessary, the domain feature is allowing non-standardization, and its use should be looked at carefully.

Sharing Attribute Values

To prevent some of these non-standardization problems, and to allow you to create smaller dictionaries that are easier to maintain, System Dictionary allows you to create entities and relationships in a local domain version and link them to existing entities or relationships of the same type in the common domain version. The linked occurrences will then share the attribute values (and the memory storage space) of the occurrences they are linked to.

For example, a Dictionary Administrator wants all last names described in the dictionary to be 32 characters long, and therefore creates an entity called LAST-NAME of type ELEMENT in the common domain version, setting the value of its attribute BYTE-LENGTH, to 32. The DA then instructs the dictionary users to use this entity for all last names. User #3, however, needs to call the entity by a different name, creates an entity called CUSTOMER-LAST-NAME and links it to the entity LAST-NAME in the common domain version. The entity CUSTOMER-LAST-NAME (the local entity) now shares the attribute value (32) of the attribute BYTE-LENGTH of the entity LAST-NAME (the common entity). Note that the names of the two linked entities do not have to be the same.

The linking feature, therefore, promotes standardization within the dictionary, and allows you to save space by using the same set of attributes for multiple entities or relationships. It also makes the dictionary simpler and easier to maintain by having only one set of attributes to modify, etc. for a set of linked occurrences, instead of several separate sets. Figure 4-4 below illustrates the common domain and one local domain, which contain common version CV1 and local version LV1 respectively. The local version LV1, which is linked to the common version CV1, contains local entity LE1 and local relationship LR1, which are linked to common entity CE1 and common relationship CR1 respectively, in the common version. Note that entities called E2 in both the common and local domains are not linked. They are therefore completely separate entities, and do not share the same attributes, even though they have the same name. Remember that you must link the local domain version to the common domain version before

you can link occurrences they contain. (Linking versions is explained on the previous page.)

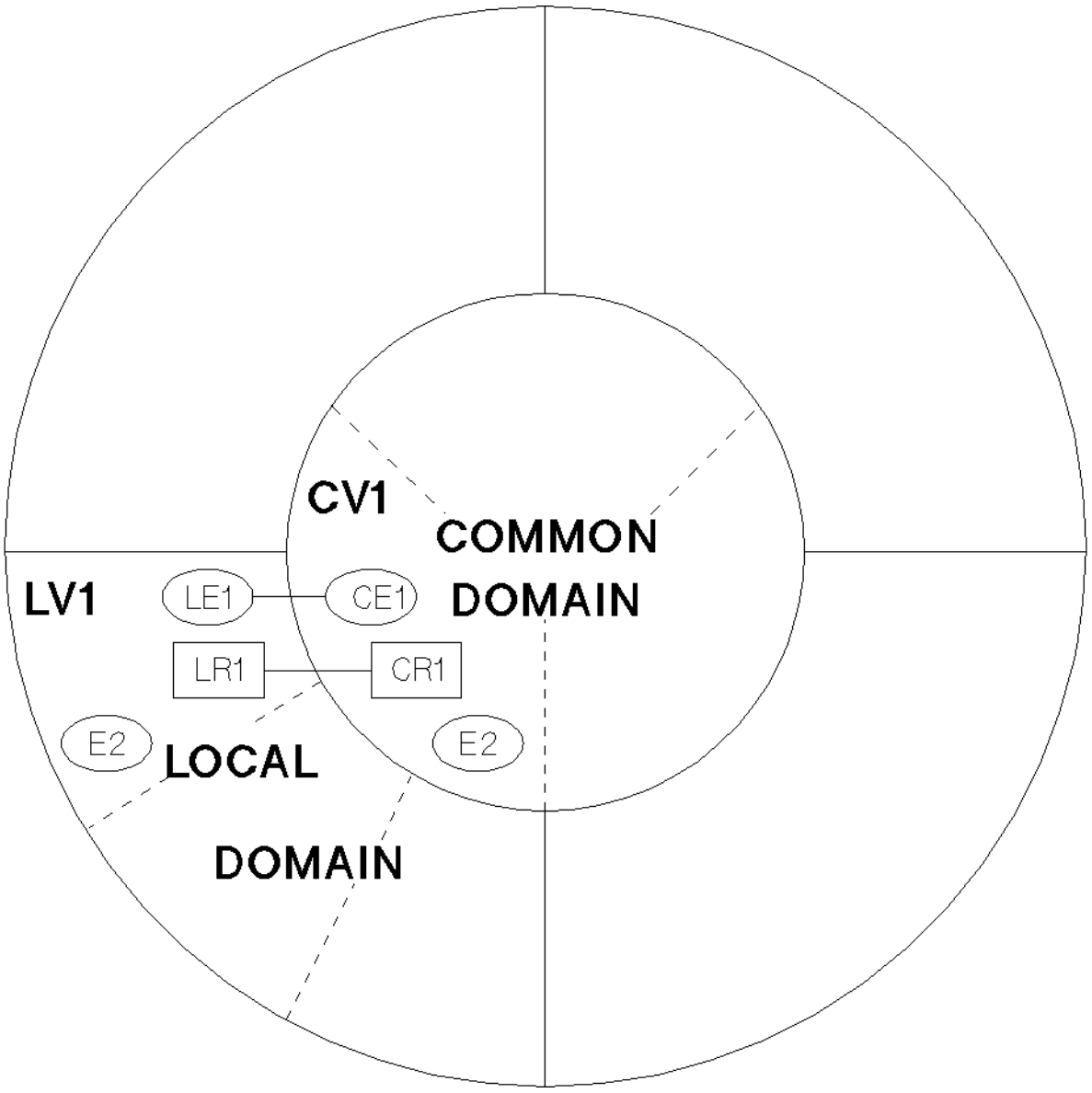


Figure 4-4. Linking Occurrences Between Domains

Special Attribute Values

A local domain occurrence shares all the attribute values of the occurrence it is linked to in the common domain, except for the values of its special attributes. (Special attributes are introduced in Chapter 3.) The values of the special attributes of the local domain occurrence are allowed to differ from those of the occurrence in the common domain, so that they will describe the linked occurrence in the local domain. This exception is necessary because System Dictionary assigns values to these attributes based on the event in which you create them.

For example, the attribute date-created, describing a linked occurrence in a local domain, contains the date on which the linked occurrence was created, not the date on which the occurrence in the common domain was created. The values of the scope-changed, and date-changed attributes are handled the same way. The values for the attributes scope-owner and sensitivity are allowed to vary so that the users of each domain have control over these security attributes. The value of the id-number attribute assigned to an

entity is arbitrary, so there is no need to keep it synchronized between occurrences. The relationship-position attribute assigns ordinal numbers to a set of relationships of the same type that have the same parent entity. The ordinal numbers implicitly define the order in which the child entities of these relationships occur within the parent entity. For example, in a series of relationships of the type RECORD contains ELEMENT that have the same RECORD entity, the relationship-position attribute implicitly defines the ordering of elements within the record. You can vary relationship-position attribute values among linked relationships so that users in each domain can re-arrange their order. The relationship CUSTOMER-ADDRESS contains LAST-NAME, for example, may be linked between domains. But in one domain, its relationship-position value may indicate that LAST-NAME comes first in the record while in another domain it may come last.

Linking Restrictions

Note that you can link entities only to entities, and you can link relationships only to relationships, and only from a local domain version to a common domain version which must be linked together first. Further, you can make links only between occurrences of the same type. For example, you cannot link an entity of type RECORD to an entity of type FILE, nor can you link a relationship of type DATABASE contains DATASET to a relationship of RECORD contains ELEMENT.

5 Dictionary Security

Overview

System Dictionary includes a security scheme to limit access to authorized users, and to control the level of access of each user. Generally, this scheme consists of the following requirements:

1. You must use the correct **scope** and its associated **password** to have the necessary access rights to the dictionary.
2. To have unlimited access to a definition or occurrence, the current scope must be either its **owner scope** or the **Dictionary Administrator scope.** A scope that does not own a particular occurrence may have explicit access to it by being **associated** with that occurrence.
3. The entities and relationships in the dictionary must have the correct **sensitivity** to allow access by a scope that is not its **owner scope**, the Dictionary Administrator scope, or a scope that is associated with the occurrence.

System Dictionary checks for the necessary **dictionary open mode** and **scope rights** it needs to succeed, and if attempting to access an occurrence, also checks the **association** and **sensitivity** of that occurrence. If any of the following occur, the operation fails and an appropriate error code is returned.

- The dictionary is not open in the proper mode.
- The current scope does not have sufficient scope rights.
- The scope attempting to access an occurrence does not have the necessary association with that occurrence.
- The sensitivity of the occurrence restricts access.

This chapter contains references to all System Dictionary *security* operations, and to a number of operations whose performance and output depend greatly on the particular security scheme installed at the time you call them. To completely understand all of the information in this chapter, it may be necessary to refer to the detailed explanations of either the System Dictionary intrinsics, which are located in Chapter 4 of the *HP System Dictionary/XL Intrinsics Reference Manual*, or the descriptions of the SDMAIN commands, which are located in Chapter 4 of the *HP System Dictionary/XL SDMAIN Reference Manual*.

Scopes

Scopes and their associated **passwords** are the primary means that System Dictionary uses to limit dictionary access to authorized users, and to control the level of access they have to the definitions that System Dictionary contains. Each scope may have up to six **scope rights** (see next page) which define the capabilities/access rights that the scope has to the dictionary.

Scopes are global in System Dictionary and are defined across structure, domains, versions, and occurrences.

Ownership

Every System Dictionary component is **owned** by a scope. System Dictionary includes a built-in scope called CORESET, which owns all of the entity types, relationship types, relationship classes, and attributes supplied on the installation tape. It also owns the **Dictionary Administrator scope**, which

either directly or indirectly owns all other scopes you create in the dictionary. All structure definitions and dictionary occurrences you create are owned by a scope. Each **owner scope** has unlimited access to and authority over the items it owns. An owner scope can transfer any or all of its owned items to another scope. It can allow other scopes to access the occurrences it owns by creating an **association** between those scopes and specified occurrences in the dictionary.

Specific information on occurrence ownership is located further on in this chapter, under the heading "Access Rights".

The Dictionary Administrator Scope

This special scope, generally referred to as the **DA scope**, has ultimate authority over and unlimited access to the entire System Dictionary. This scope exists after you initialize System Dictionary. It is owned by the core set, and you can never delete it. The DA scope has all of the scope rights listed on the next page, which, in this scope, you can never delete or modify. The DA scope is the only scope that can change the DA password and DA scope external name. It can also call any System Dictionary intrinsic.

NOTE The DA scope and its password are considered highly sensitive and privileged information.

The DA scope may create other scopes (which it then **owns**) and may assign them individual scope rights. If they have the *Secure* scope right, those scopes may, in turn, create other scopes (which *they* own) and may assign to them, any of their own scope rights. Although there is no limit to the number of nested levels of scopes in the dictionary, the maximum number of scopes that System Dictionary allows is 128.

Scope Rights

- **Scope rights** are capabilities associated with a scope. Scope rights specify the dictionary components that the scope is allowed to manipulate. They also specify the dictionary operations that a particular scope is allowed to perform. The six scope rights are:
- **Secure** capability indicates that a scope has the ability to create and own other scopes, and retrieve information about security.
- **Extend** capability indicates that you have the ability to extend and customize the dictionary by creating and owning new entity types, relationship types, relationship classes, and attributes.
- **Create** capability indicates that a scope has the ability to create and own entities and relationships.
- **Read** capability indicates that a scope can read entities and relationships accessible to that scope. Note that if a scope has *create* capability, *read* capability is automatically assigned.
- **Domain** capability indicates that you can create and own dictionary domains, and retrieve information about domains.
- **Version** capability indicates that you can create or own versions and perform version control operations. Note that if a scope has *domain* capability, *version* capability is automatically assigned.

Scope Right Restrictions There are some restrictions on modifying scope rights. Generally, System Dictionary does not allow any scope right to be removed from a scope if that scope right is currently in use. For example:

- A scope cannot be modified to remove *secure* capability if that scope owns any scopes.
- A scope cannot be modified to remove *extend* capability if that scope owns any attributes, entity types, relationship types, relationship classes, or type/attribute associations.
- A scope cannot be modified to remove *create* capability if that scope owns any entity or relationship occurrences.
- A scope cannot be modified to remove *domain* capability if that scope owns any domains.
- A scope cannot be modified to remove *version* capability if that scope owns any versions.

Scope Restrictions

The following restrictions apply to scopes.

- Only the DA scope or a scope with *secure* capability can create a new scope.
- When a scope creates a new scope, it becomes the **owner** of the new scope.
- A scope cannot assign a scope right that it does not have itself.
- Only the DA scope or the owner of a scope can modify, delete, or retrieve information about that scope. Note, however, that a scope can change its own password, and retrieve information about itself.
- When a scope is deleted, an existing scope may be specified to become the new owner of everything that is owned by the scope to be deleted. The new owner scope must, however, have at least the same scope rights as the scope it replaces. If a scope is not specified, the current scope will become the new owner, but only if it has at least the same scope rights as the scope it replaces.
- Only the DA scope can retrieve all the scopes in the dictionary, but a scope with *secure* capability can retrieve all of the scopes it owns.

Scope Password

Each scope in System Dictionary has an associated password. A password is necessary to gain access to a particular scope. Scope passwords are **case sensitive**.

Using Scopes

System Dictionary allows five different operations to be performed directly on scopes. There are also a number of dictionary operations that use scopes, these are covered further on in this chapter. The five operations are:

- Creating scopes.
- Deleting scopes.
- Modifying scopes (can change its own password or that of another scope it owns, or change that scope's name, scope rights, or scope owner).
- Retrieving information about a specified scope.
- Retrieving a list of all the scopes in the dictionary, or all that the scope owns.

Structure Security

The following groups of restrictions are placed by System Dictionary on the operations associated with the dictionary structure. Each group contains the restrictions for a single component type in the dictionary structure. Although the groups contain similar restrictions, they are listed separately for ease of use.

Entity Type Restrictions

System Dictionary provides the following security for entity types:

- Only the DA scope or a scope with extend capability is allowed to create new entity types.
- When a scope creates a new entity type, it becomes the owner of that entity type.
- Only the DA scope or the owner scope can delete or rename an entity type, or change its owner scope.
- Core set entity types are owned by the core set and can never be deleted or modified (exception: core set entity type external names may be modified by the DA scope.)

Relationship Type Restrictions

System Dictionary provides the following security for relationship types:

- Only the DA scope or a scope with extend capability is allowed to create new relationship types.
- When a scope creates a new relationship type, it becomes the owner of that relationship type.
- Only the DA scope or the owner scope can delete a relationship type or change its owner scope.
- Core set relationship types are owned by the core set and can never be deleted or modified.

Relationship Class Restrictions

System Dictionary provides the following security for relationship classes.

- Only the DA scope or a scope with extend capability is allowed to create new relationship classes.
- When a scope creates a new relationship class, it becomes the owner of that relationship class.
- Only the DA scope or the owner scope can delete or rename a relationship class or change its owner scope.
- Core set relationship classes are owned by the core set, and can never be deleted or modified (exception: core set relationship class external names may be modified by the DA scope).

Attribute Restrictions

System Dictionary provides the following security for attributes:

- Only the DA scope or a scope with extend capability is allowed to create new attributes, or add an attribute to either an entity type's attribute list or to a relationship type's attribute list.
- When a scope creates a new attribute, it becomes the attribute's owner. When a scope adds an attribute to an attribute list, it becomes the owner of that entity type/attribute pair or relationship type/attribute pair.
- Only the DA scope or the owner scope can delete or rename an attribute or change its owner scope, length, or edit values.
- Only the DA scope or the scope that owns the pair may delete an attribute from an entity type's attribute list or from a relationship type's attribute list.
- Core set attributes and attribute pairs are owned by the core set and can never be deleted or modified. (exception: core set attribute external names and edit value lists may be modified).

Domain and Version Security

This subsection describes the security scheme for System Dictionary domains and the versions of the dictionary occurrences within each domain. The occurrences also have protection, which is discussed further on in this chapter. (Descriptions of domains and versions are located in Chapter 4 of this manual.)

Domain Security

Domain security controls which scopes have access to which domains. Therefore, the current scope must have access to the specified domain when opening the dictionary or switching domains. Further, when switching scopes, the new scope must have access to the current domain. The security for any domain depends on:

- The access rights of the current scope to that domain.
- The sensitivity of the domain.

Access Rights.

The access rights of a scope to a domain are determined by whether the scope owns the domain or is just associated with it. Association and ownership are discussed below.

DOMAIN OWNERSHIP

When a scope owns a domain, it has all rights to that domain, and can therefore modify it, transfer its ownership to another scope, or even delete it. It can also allow another scope access to the domain by associating the domain with that scope. Note that the DA scope always has all rights to all domains.

The security of a domain applies indirectly to all versions within that domain. Although the version itself does not have security, the current scope must have access to the domain containing the version it is trying to access.

DOMAIN/SCOPE ASSOCIATION

An association between a domain and a scope is an explicit access capability granted to that scope by the owner scope of that domain. However, even though a scope has access to a given domain, it cannot do operations within that domain (create, access, or delete occurrences, for example) unless it also has the necessary scope rights for those operations.

A scope can delete domain associations it has created from any domain/scope association. It can also delete domain/scope associations from itself.

Sensitivity.

The security of a domain is actually set by its Sensitivity. Domain sensitivity is set to one of two values when you create a domain, and can be changed only by its owner scope or the DA scope. The two values are:

1 1. = Private sensitivity: Only the DA scope or the scope that owns the domain is allowed access to it, unless the DA scope or owner scope assigns access to other scopes through domain/scope associations.

2 2. = Public sensitivity: Any scope may access the domain.

When using the intrinsics to create a domain, you must specify the sensitivity of that domain, as no default exists except when using SDMAIN. Note that the sensitivity of a domain should be carefully determined. If you change the sensitivity from public to private, all scopes that previously had access to this domain will no longer have access, unless that domain is explicitly associated with them.

NOTE The sensitivity of the common domain is set to public, and cannot be modified.

Domain Restrictions

System Dictionary provides the following security for domains.

- Only the DA scope or a scope with domain capability is allowed to create new domains.
- When a scope creates a new domain, it becomes the owner of that domain.
- Only the DA scope or the owner scope can delete or rename a domain or change its owner scope.

Version Restrictions

System Dictionary provides the following security for versions:

- Only the DA scope or a scope with version capability is allowed to create new versions.
- When a scope creates a new version, it becomes the owner of that version.
- Only the DA scope or the owner scope can delete or rename a version or change its owner scope.
- Only the DA scope or a scope with version capability can set the status of a version, or copy all occurrences of an existing version and assign them to a new version.

Note that when a version is copied, all occurrences of the existing version are copied to another version. The owner for each occurrence will be the same across all versions.

Occurrence Security

The security scheme for entities is exactly the same as that for relationships, and is accomplished by two parallel sets of operations, one for entities and one for relationships. In the following paragraphs, therefore, occurrence can mean either entity or relationship.

Figure 5-1 outlines the security scheme for occurrences. Note that the boxed text in the sections labeled 'Scope Capability' and 'Linked Scope Access' lists the capability of the scope. The text in parentheses shows the actual access the scope has to an occurrence. This access is determined by the combination of occurrence sensitivity and scope access rights, which are explained in detail on the following pages, and illustrated below.

In the example, note that a scope having create capability can read or modify an occurrence whose sensitivity is set to public modify, but cannot modify an occurrence whose sensitivity is set to public read, unless that scope is either the owner of the occurrence or has been given modify access to the occurrence by its owner.

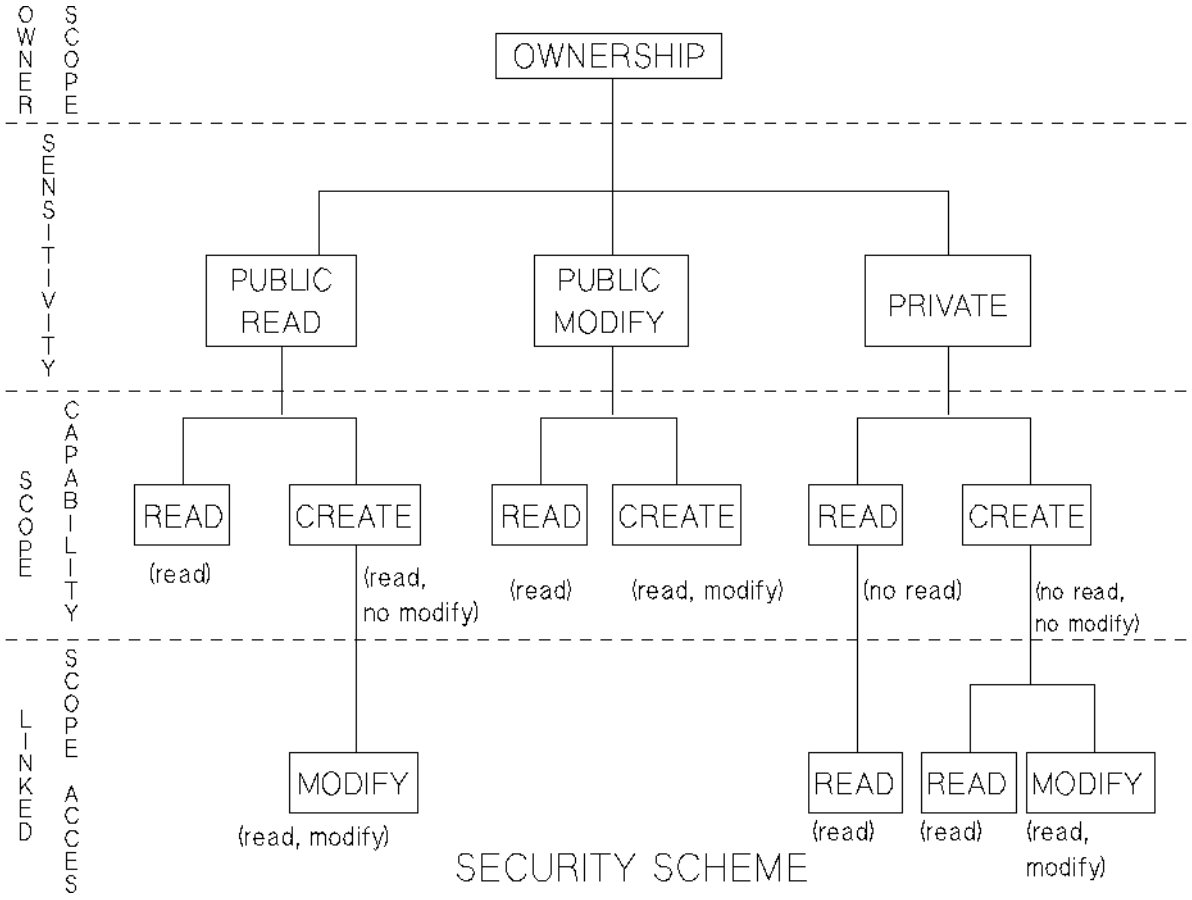


Figure 5-1. Occurrence Security

The security for any occurrence depends on the following:

* The access rights of the current scope to that occurrence.
* The sensitivity of the occurrence.

Access Rights

The access rights of a scope to an occurrence are determined by whether the scope owns the occurrence or is just associated with it. Association and ownership are discussed below.

Occurrence Ownership.

When a scope owns an occurrence, it has all rights to that occurrence, and can therefore read it, modify it, transfer its ownership to another scope, or even delete it. It can also allow another scope access to the occurrence by associating the occurrence with that scope. Note that the DA scope always has all rights to all occurrences.

The security of an occurrence applies to all versions of that occurrence. When you create an occurrence, if no other version of that occurrence exists, the current scope becomes the owner scope. If another version of that occurrence already exists, then the current scope must be the DA scope or the owner scope of the existing version of the occurrence.

Occurrence/Scope Association.

An association between an entity or relationship occurrence and a scope is an explicit access capability granted to that scope by the owner scope of that occurrence. When a scope is associated with an occurrence, its rights to that occurrence are dependent upon the type of association it has. Associations between scopes and occurrences can be either of two types:

- Read Access, which gives the associated scope the capability to read the occurrence. Further information about retrieving information about occurrences is located in Chapter 4 of the HP System Dictionary/XL Intrinsics Reference Manual. Refer to the intrinsics SDGetEnt, SDGetEntList, SDGetRel, and SDGetRelList, which are used to retrieve information about occurrences.
- Modify Access, which not only gives the associated scope the capability to read the occurrence, but also allows it to change the values of most of its attributes.
  To modify an occurrence, the associated scope must also have create capability, and if creating links, must also have read access to the common domain entity or common domain relationship involved in the link.

When an occurrence is associated to a scope, all versions of that occurrence are associated with the scope.

A scope can delete occurrence associations it has created from any occurrence/scope association. It can also delete occurrence/scope associations from itself. For a local domain occurrence to be linked to a common domain occurrence, the owner scope must have at least read access to the common domain occurrence. Only the owner scope or DA scope can modify the link to a common domain occurrence. Note that a scope that has access to a common domain occurrence can also read the names of all of the local domain occurrences that are linked to that common domain occurrence.

A two step process is required to modify an occurrence/scope association (change the association between an occurrence and a scope).

- Delete the current association.
- Create a new association.

Further details on associations are located in the descriptions of intrinsics SDAddEntScope, SDAddRelScope, SDDeleteEntScope and SDDeleteRelScope in Chapter 4 of the HP System Dictionary/XL Intrinsics Reference Manual.

Sensitivity

Sensitivity is an attribute of an occurrence. It is set to one of three values when you create the occurrence, and can be changed only by its owner scope or the DA scope. The three values are:

1. 0 = Private sensitivity: Only the DA scope or the scope that owns the occurrence is allowed access to it, unless the scope assigns access to other scopes through occurrence/scope associations.
2. 1 = Public Read sensitivity: Any scope with at least read capability may read the occurrence. The DA scope or owner scope can assign other scopes modify access to the occurrence through occurrence/scope associations.
3. 2 = Public Modify sensitivity: Any scope with read capability may read the occurrence. Any scope with at least create capability may read and modify the occurrence.

You should determine the sensitivity of an occurrence carefully. If you do not specify the sensitivity when you create the occurrence, it defaults to the value specified in its attribute edit, or to private, if no attribute edit exists. If you change the sensitivity from public to private, all scopes that previously had access to this occurrence will no longer have access, unless that occurrence is explicitly associated with them.

Specific Restrictions

System Dictionary provides security that is specific to entities and security that is specific to relationships. These are discussed here.

Only the DA scope or the owner scope can create or delete a synonym for an entity. Any scope with at least read access to an entity can list all of an entity's synonyms.

Only the DA scope or the owner scope can change the scope-owner and sensitivity attribute values of an occurrence, even if other scopes have modify access to the occurrence.

NOTE If a scope deletes an entity it owns, then all relationships involving that entity are also deleted, even if the scope does not own or have access to all those relationships.

Example of Entity Security

The next two pages provide an example of how you could set up security for specific scopes and entities in System Dictionary, and then modify them to increase the access of each scope.

You define Scope1 and Scope2 in the dictionary with create and read capability. You define Scope3 in the dictionary with read capability. A user opens the dictionary with the scope Scope1. This user creates the file File1 and the record Record1, both with sensitivity set to private, and element Element1 with sensitivity set to public modify. Scope1 is the owner of these three entities. Only Scope1 or the DA scope can delete these entities.

Another user opens the dictionary with Scope2. This user creates element Element2 with sensitivity set to public read and element Element3 with sensitivity set to private. Scope2 is the owner of these entities. Only Scope2 or the DA scope can delete these entities.

The table below lists each of the entities the scopes have access to at this point. Note that neither Scope2 nor Scope3 have access to File 1 or Record 1, and that Scope3 has at most have read access to an entity

because it has just read capability.

Table 6: Security Example

SCOPE	ENTITIES	ACCESS CAPABILITY
Scope1 (create, read)	File1 Record1 Element1 Element2	modify, delete, read modify, delete, read modify, delete, read read
Scope2 (create, read)	Element1 Element2 Element3	modify, read modify, delete, read modify, delete, read
Scope3 (read)	Element1 Element2	read read

To allow Scope2 and Scope3 access to File1 and Record1, Scope1 associates these entities to those scopes. Both Scope2 and Scope3 are given read ScopeAccess to Record1, Scope2 is given modify ScopeAccess to File1, and Scope3 is given read ScopeAccess to File1.

The table below lists each of the entities the scopes have access to after these changes were made. Note the addition of File 1 and Record 1 to Scope2 and Scope3.

Table 7: Security Example After Modifying Scopes

SCOPE	ENTITIES	ACCESS CAPABILITY
Scope1	File1 Record1 Element1 Element2	modify, delete, read modify, delete, read modify, delete, read read
Scope2	Element1 Element2 Element3 File1 Record1	modify, read modify, delete, read modify, delete, read modify, read read
Scope3	Element1 Element2 File1 Record1	read read read read

To create a relationship between File1 and Record1, the scope must be either the DA scope, Scope1, or a scope with create capability and at least read access to File1 and Record1 which includes Scope2 but not Scope3. Again, Scope3 is only able to read from the dictionary because it has just read capability. If Scope2 creates a relationship between File1 and Record1 then Scope2 becomes the owner of that relationship. Only Scope2 or the DA scope can delete that relationship. Only Scope2, the DA scope or a scope with modify access to that relationship can modify that relationship.

6 The Core Set

Overview

This chapter describes the **core set** of definitions provided with System Dictionary, and shows how this set supports various HP subsystems.

The Core Set

System Dictionary contains a number of predefined entity types, relationship types, relationship classes, and attributes, known as the **core set**. These structures are created when the dictionary is initialized, and are owned by a scope called CORESET. (Ownership is discussed in Chapter 5 of this manual.) The purpose of the core set is to provide a standard base of definitions for you, and to support the description of the following HP subsystems:

- MPE files
- MPE Accounting structure
- IMAGE and TurboIMAGE DBMS
- VPLUS forms
- RAPID/V
- Network node locations
- COBOL data definitions
- Pascal data definitions (subset of Pascal data types)
- RPG programs

Modifying the Core Set

Only the Dictionary Administrator scope can modify the core set. Only limited modifications can be made, as listed below.

- Change the external names of entity types, relationship classes, and attributes.
- Modify attribute edits of core set attributes.
- Add attributes to core set entity types or relationship types. Note that although the *individual* components are owned by the core set, the entity type/attribute pair or relationship type/attribute pair is owned by the scope that created it.

Extending the Dictionary Structure

System Dictionary is designed to be a repository for resource definitions. However, because many types of definitions may exist within your resources, it is necessary to customize the dictionary to fit your environment. The primary means of customizing System Dictionary is to *extend* the dictionary structure. Operations that extend the dictionary structure include the following:

- Adding attributes to core set entity types or relationship types.Note that although the *individual* components are owned by the core set, the entity type/attribute pair or relationship type/attribute pair is owned by the scope that created it.
- Adding entity types, relationship types, relationship classes, and attributes that you created.

- Adding attributes to an entity type or relationship type that you created.

You can use these operations to create a custom structure that closely models your resource environment.

CAUTION The core set is included in System Dictionary to provide a standard set of definitions. Extending this set should be carefully planned, as extension may cause the loss of standardization between dictionaries.

The process to extend the dictionary structure is actually quite simple, as shown below.

1 Open the dictionary in *exclusive customization* mode with a scope that has *extend* capability and the necessary access rights to any definitions being modified or deleted.

2 Use the appropriate SDMAIN commands or dictionary intrinsics to add, modify, or delete the desired structure definitions.

3 Close the dictionary.

The Extended Set.

The set of definitions you create that extends the core set is called the **extended set**, and you can access it in exactly the same way as the core set. The only real difference between the core set and the extended set is that you can modify and delete the components in the extended set, while you essentially cannot modify or delete the components in the core set.

Restructuring the Dictionary

A process called **restructuring** is necessary to include any structure changes into the working dictionary. This process is similar to compiling a program, and occurs automatically whenever the dictionary is closed from the *exclusive customization* mode. Refer to Chapter 3 for more information on restructuring the dictionary.

Core Set Attributes

The following attributes are included in the System Dictionary core set:

access This attribute is a 2-character field that describes the IMAGE access rights. Valid values are ' ', 'R' and 'W'.

back-reference-flag Attributes that are associated to a relationship type and are also associated to the second entity type of that relationship type are called corresponding attributes (excluding the special attributes). This attribute is a Boolean field that indicates where the relationship's corresponding attribute values reside. If false, the corresponding attribute values reside with the relationship. If true, then the corresponding attributes values reside with the definition of the second entity of the relationship.

blank This attribute is a Boolean field that describes the display option on an element entity. If true, then an element entity whose value is zero is displayed as a blank character string.

blocking-factor This attribute is a 2-byte binary field that describes the blocking factor a data set entity uses.

blocking-max This attribute is a 2-byte binary field that describes the maximum blocking factor a file entity uses.

blocking-min This attribute is a 2-byte binary field that describes the minimum blocking factor a file

entity uses.

blocking-units This attribute is a 10-character field that describes the blocking factor units a file entity uses. The valid values are 'RECORDS' or 'CHARACTERS'.

byte-length This attribute is a 4-byte binary field that describes the number of bytes you need to store an entity.

byte-offset This attribute is a 4-byte binary field that describes the byte offset of a child record or element within a parent record.

capacity This attribute is a 4-byte binary that field describes the capacity of an IMAGE data set.

category-type This attribute is a 10-character field that contains the category type.

cctl-flag This attribute is a Boolean field that describes carriage control for a file entity. If true, then the file has carriage control; if false, then no carriage control.

char-type This attribute is a 10-character field that describes the character set that a file uses. The valid values are 'ASCII' or 'EBCDIC'.

class-number This attribute is a 4-byte binary field that contains the IMAGE security class number.

cobol-alias This alias attribute is a name field that contains aliases for COBOL source declaration.

constant-type This attribute is a 2-character field that describes the type of a constant. The possible values are:

U = upper case ASCII

X = upper or lower case ASCII

9 = numeric ASCII

Z = zoned decimal

P = packed decimal

I = integer binary

J = integer binary (COBOL)

K = logical

R = floating point (commercial)

E = floating point (E format)

B = Boolean

S = Pascal string

D = Basic Business decimal

count This attribute is a 4-byte binary field that specifies the number of elements in an array.

date-created This attribute is a 16-character field that describes the date and time an entity or relationship was created. The dictionary intrinsics always generates its value. The for-

mat of the attribute is: YYYYMMDDhhmmsstt where

YYYY is the year

MM is the month

DD is the day

hh is the hour

mm is the minute

ss is the seconds

tt is the tenths of seconds

date-changed This attribute is a 16-character field that describes the date and time the entity or relationship was last changed. The dictionary intrinsics always generate its value. The format of the attribute is the same as that described in the **date-created** attribute.

decimal This attribute is a 4-byte binary field that describes the number of places to the right of the decimal point of an element.

default This attribute is a variable length field that contains a default value.

description This attribute is a variable length field that holds description text.

directive This attribute is a 10-character field that describes a compilation directive.

directory-version This attribute is a 2-byte binary field that indicates the version of the Network Directory Path Report.

display-length This attribute is 4-byte binary field that contains a count of the number of display characters or digits of an entity.

document-type This attribute is a 10-character field that contains the document type. Example document types are 'CHART', 'DRAW', 'FIGURE'.

edit-mask This attribute is a variable length field that contains an edit mask used for displaying or inputting an entity's value.

element-display This attribute is a Boolean field that flags whether an element entity should be displayed on the Inform Data Names Menu. If true, it is displayed. If false, it is not displayed.

element-type This attribute is a 2-character field that describes the type of an element. The possible values are:

U = upper case ASCII

X = upper or lower case ASCII

9 = numeric ASCII

Z = zoned decimal

P = packed decimal

I = integer binary

J = integer binary (COBOL)

K = logical

R = floating point (commercial)

E = floating point (E format)

B = Boolean

S = Pascal string

* = back references

D = Basic Business decimal

A numeric type followed by a '+' indicates positive values only.

entity-long-name This attribute is a variable length field that stores the descriptive long name of an entity.

entry-text This attribute is a variable length field that contains a prompt string for an entity.

field-enhancement This attribute is a 4-character field that describes the display enhancement for a VPLUS form field. Some combination of up to 4 of the following codes may be used:

H = half bright

I = inverse video

U = underline

B = blink

S = security

1-8 = color

NONE = none

field-number This attribute is a 2-byte binary field that contains the VPLUS form field number.

field-type This attribute is a 2-character field that describes the level of requirement of a VPLUS field. Possible values are:

D = display

R = required

O = optional

P = process

file-dev-class This attribute is a 2-character field that describes the class of device on which a file resides, used by COBOL. Possible values are:

DA = ss storage device

UT = utility device (such as tape drive)

UR = unit record device (such as a card reader)

file-size This attribute is a 4-byte binary field that specifies the maximum file capacity. The units depend on the **record-format**: for FIXED length record-formats the file-size is in terms of logical records; for VARIABLE or UNDEFINED record-formats the file-

size is in terms of blocks.

file-type This attribute is a 10-character field that describes the access method of the file. Possible values are 'SEQUENTIAL', 'RELATIVE', and 'LOG' for log files.

function-result This attribute is a 32-character field that describes the resulting type of a function subroutine module entity.

head-form This attribute is a Boolean field to flag that a form is the head form of a formsfile. If true, the form is the head form.

heading-text This attribute is a variable length field that contains a report heading for an entity.

hp-condition-value This attribute is a variable length field that holds all values and ranges to be assigned to a COBOL condition-name.

hpdbe-file-type This attribute is a 10 character field that describes the DBEFile type. Valid values are 'MIXED', 'INDEX', and 'TABLE'.

hpdbe-log-buffers This attribute is a 2 byte binary field that defines the number of log buffers a DBEnvironment uses.

hpdbe-max-transactions This attribute is a 2 byte binary field that defines the maximum number of concurrent transactions a DBEnvironment supports.

hpdbe-multi-users This attribute is a Boolean field that indicates whether a DBEnvironment allows multiple user access. If TRUE, multiple user access is allowed. If FALSE, only single user access is allowed.

hpdbe-page-buffers This attribute is a 2 byte binary field that defines the number of page buffers a DBEnvironment uses.

hpdbe-pages This attribute is a 4 byte binary field that defines the number of pages in a DBEnvironment file.

hpsql-alias This attribute is a name field that contains aliases for HP SQL access.

hpsql-alter-auth This attribute is a Boolean field that indicates whether alter authority has been granted. If true, alter authority has been granted.

hpsql-auth-name-type This attribute is a 10-character field that describes the HP SQL authorization name type. Valid values are 'AUTH-GROUP' and 'USER'.

hpsql-clustering This attribute is a Boolean field that describes whether clustering has been specified for an HP SQL index within an HP SQL table. If true, clustering has been specified.

hpsql-connect-auth This attribute is a Boolean field that indicates whether connect authority has been granted. If true, connect authority has been granted.

hpsql-dba-auth This attribute is a Boolean field that indicates whether dba authority has been granted. If true, dba authority has been granted.

hpsql-delete-auth This attribute is a Boolean field that indicates whether delete authority has been granted. If true, delete authority has been granted.

hpsql-index-auth This attribute is a Boolean field that indicates whether index authority has been granted. If true, index authority has been granted.

hpsql-insert-auth This attribute is a Boolean field that indicates whether insert authority has been

granted. If true, insert authority has been granted.

hpsql-lock-mode This attribute is a 10-character field that contains the lock mode of an HP SQL table. Valid values are 'PUBLIC', 'PUBLICREAD' and 'PRIVATE'.

hpsql-resource-auth This attribute is a Boolean field that indicates whether resource authority has been granted. If true, resource authority has been granted.

hpsql-select-auth This attribute is a Boolean field that indicates whether select authority has been granted. If true, select authority has been granted.

hpsql-select-command This attribute is a variable length field that contains an HP SQL select command.

hpsql-update-auth This attribute is a Boolean field that indicates whether update authority has been granted. If true, update authority has been granted.

id-number This attribute is a 4-byte binary field that contains a numeric identifier for an entity. Its value is not generated by the dictionary intrinsics and it is not checked for uniqueness.

image-alias This alias attribute is a name field that contains aliases for IMAGE database accessing.

image-class-type This attribute is a 10-character field that contains the security class type.

image-database-type This attribute is a 10-character field that describes the database type. Possible values are 'IMAGE', 'HPIMAGE', or 'TURBO'.

image-dataset-type This attribute is a 10-character field that describes the data set type. Possible values are 'MANUAL', 'AUTOMATIC', 'DETAIL', or 'RELATION'.

inform-group-type This attribute is a 10-character field that contains the group type.

justify This is a Boolean field that describes the justification an element entity uses. If true, the alphanumeric element entity should be right justified.

ksamfile-type This attribute is a 10-character field that describes the type of the KSAM file. Possible values are 'DATA' and 'KEY'.

language This attribute is a 10-character field that describes the programming language that a module or program is written in.

ldev-number This attribute is a 2-byte binary field that contains the logical device number.

link-value This attribute is 2-byte binary field that contains the link priority of an element entity. Valid values are:

-1 = the element cannot be used for linking

0 = the element may or may not be used for linking

1 or more = the element should be used as a link when possible; the lower the positive integer the higher the priority

lockword This attribute is an 8-character field that contains a lockword.

max-record-size This attribute is a 4-byte binary field that describes the maximum record size of a file entity in bytes.

min-record-size This attribute is a 4-byte binary field that describes the minimum record size of a file entity in bytes.

module-type This attribute is a 10-character field that contains the module type.

node-name-changed This attribute is a 50-character field that contains the NODE NAME which de the last change to any NODE NAME.

node-name-type This attribute is a 2-byte binary field that contains the NODE NAME type. Valid value is: 1.

not-null This attribute is a Boolean field that indicates whether an element cannot have null values. If true, the element cannot have null values.

parm-flag This is a Boolean field that describes that the element is a parameter of a module.

parm-number This is a 4-byte binary field that describes the position of a parameter in a module.

pascal-alias This alias attribute is a name field that contains aliases for Pascal source declaration.

pass-method This is a 10-character field that describes the method in which a parameter is passed to a module. Valid values are 'REFERENCE', 'VALUE' and 'READONLY'.

password This attribute is a 16-character field that contains a password used with an IMAGE security class entity.

path-report This attribute is a variable length field that contains the path report of a NODE NAME.

primary-flag This attribute is a Boolean field that describes the primary key in a KSAM entity type or an IMAGE chain.

primary- record This attribute is a Boolean field that describes the primary record in a data set, file, KSAM file, table, or record.

record-fort This attribute is a 10-character field that describes the file record type. Valid values are 'FIXED', 'VARIABLE', 'UNDEFINED' and 'SPANNED'.

recording- mode This attribute is a 10-character field that describes the file type: 'ASCII' or 'BINARY'.

relationship-position This attribute is a 4-byte binary field that describes the relationship position of an entity (for example, first child, second child, etc.). The value must be a unique positive integer. A zero causes the dictionary intrinsics to generate a default value greater than the current largest value for the relationship position.

right-justify This attribute is a Boolean field that describes the type of justification the element entity value uses; if true then the value of the element entity should be right justified.

scope-changed This attribute is a 32-character field that contains the name of the scope which last modified an entity or relationship. Its value is always generated by the dictionary intrinsics.

scope-owner This attribute is a 32-character field that contains the scope owning the entity or relationship. Its value is always generated by the dictionary intrinsics. Detailed information on scopes and scope ownership is located in Chapter 4 of this manual.

sensitivity This attribute is a 2-byte binary field that describes the accessibility of an occurrence as follows:

0 = Private; only the scope that owns the occurrence is allowed access, unless it assigns access to other scopes with occurrence/scope associations.

1 = Public Read; any scope may read the occurrence. The owner scope can assign other scopes modify access to the occurrence with occurrence/scope associations.

2 = Public Modify; any scope may read or modify the occurrence. More information on sensitivity is located in Chapter 4 of this manual.

sign This attribute is a 2-character field that contains the sign position of a numeric element entity:

blank = no sign

LO = leading overpunch

LS = leading separate

TO = trailing overpunch

TS = trailing separate

standard-alias This alias attribute is a name field that contains aliases for general purposes.

synchronize This is a Boolean field that describes the storage alignment an element entity uses. If true, the element entity should always be aligned on word boundaries.

system-type This attribute is a 10-character field that contains the system type.

unique This is a Boolean field that describes that a KSAM file key or a table index must be unique.

units This attribute is a 10-character field that specifies the unit of measure of an entity.

vplus-alias This alias attribute is a name field that contains aliases for VPLUS formsfile accessing.

Core Set Entity Types

The following entity types are included in the System Dictionary core set.

Table 8: Core Set Entity Types

Entity Type	Description	Core Set Attributes
CATEGORY	Describes a Dictionary/3000 category.	category-type scope-owner scope-changed date-created date-changed sensitivity id-number

Table 8: Core Set Entity Types

Entity Type	Description	Core Set Attributes
CONSTANT	Describes a constant value used in programs or modules.	constant-type scope-owner scope-changed date-created date-changed sensitivity id-number
COPYLIB	Describes a COBOL copylib.	scope-owner scope-changed date-created date-changed sensitivity id-number
DEVICE	Describes a device on a node or in the network.	ldev-number scope-owner scope-changed date-created date-changed sensitivity id-number
DEVICE-CLASS	Describes a class of devices on a node or network.	scope-owner scope-changed date-created date-changed sensitivity id-number
DICTIONARY	Describes a System Dictionary.	scope-owner scope-changed date-created date-changed sensitivity id-number
DICTIONARY-REPORT	Describes a System Dictionary report	scope-owner scope-changed date-created date-changed sensitivity id-number

Table 8: Core Set Entity Types

Entity Type	Description	Core Set Attributes
DOCUMENT	Describes a document.	scope-owner scope-changed date-created date-changed sensitivity id-number document-type
ELEMENT	Describes a data item used in modules, programs, datasets, tables, KSAM files, formfiles, etc.	scope-owner scope-changed date-created date-changed sensitivity id-number display-length decimal byte-length count units sign blank justify synchronize element-type
FILE	Describes a sequential or relative file.	file-type scope-owner scope-changed date-created date-changed sensitivity id-number lockword file-size blocking-units blocking-min blocking-max char-type record-format recording-mode min-record-size max-record-size file-dev-class

Table 8: Core Set Entity Types

Entity Type	Description	Core Set Attributes
FORM	Describes a VPLUS form.	scope-owner scope-changed date-created date-changed sensitivity id-number
FORMSFILE	Describes a VPLUS/3000 forms file	scope-owner scope-changed date-created date-changed sensitivity id-number lockword
HP-CONDITION-NAME	Describes a COBOL condition-name used in a LEVEL-88 description	scope-owner scope-changed date-created date-changed sensitivity id-number
HPDBE-FILE	Describes a DBEnvironment DBEFile.	scope-owner scope-changed date-created date-changed sensitivity id-number hpdbe-file-type hpdbe-pages
HPDBE-FILESET	Describes a DBEnvironment DBEFileSet.	scope-owner scope-changed date-created date-changed sensitivity id-number
HPDBE-LOGFILE	Describes a DBEnvironment logfile.	scope-owner scope-changed date-created date-changed sensitivity id-number hpdbe-pages

Table 8: Core Set Entity Types

Entity Type	Description	Core Set Attributes
HPDBENVIRONMENT	Describes an HP SQL DBEnvironment	scope-owner scope-changed date-created date-changed sensitivity id-number hpdbe-multi-users hpdbe-page-buffers hpdbe-log-buffers hpdbe-max-transactions
HPSQL-AUTH-NAME	Describes an HP SQL authorization group or user.	scope-owner scope-changed date-created date-changed sensitivity id-number hpsql-auth-name-type hpsql-connect-auth hpsql-dba-auth hpsql-resource-auth
HPSQL-INDEX	Describes an HP SQL index.	scope-owner scope-changed date-created date-changed sensitivity id-number
HPSQL-TABLE	Describes an HP SQL table.	scope-owner scope-changed date-created date-changed sensitivity id-number hpsql-lock-mode
HPSQL-VIEW	Describes an HP SQL view.	scope-owner scope-changed date-created date-changed sensitivity id-number

Table 8: Core Set Entity Types

Entity Type	Description	Core Set Attributes
IMAGE-CLASS	Describes an IMAGE security class	image-class-type scope-owner scope-changed date-created date-changed sensitivity id-number password class-number
IMAGE-DATABASE	Describes an IMAGE database.	image-database-type scope-owner scope-changed date-created date-changed sensitivity id-number
IMAGE-DATASET	Describes an IMAGE data set.	image-dataset-type scope-owner scope-changed date-created date-changed sensitivity id-number
INFORM-CLASS	Describes an Inform/3000 security class.	scope-owner scope-changed date-created date-changed sensitivity id-number password class-number
INFORM-GROUP	Describes an Inform/3000 group.	inform-group-type scope-owner scope-changed date-created date-changed sensitivity id-number

Table 8: Core Set Entity Types

Entity Type	Description	Core Set Attributes
INFORM-REPORT	Describes a report.	scope-owner scope-changed date-created date-changed sensitivity id-number
KSAMFILE	Describes a KSAM file.	ksamfile-type scope-owner scope-changed date-created date-changed sensitivity id-number lockword blocking-units blocking-min blocking-max record-format min-record-size max-record-size file-size
LOCATION	Describes the location of a data entity.	scope-owner scope-changed date-created date-changed sensitivity id-number
MODULE	Describes a module, procedure, function or program	module-type scope-owner scope-changed date-created date-changed sensitivity id-number language function-result
MPE-ACCOUNT	Describes an MPE account.	scope-owner scope-changed date-created date-changed sensitivity id-number

Table 8: Core Set Entity Types

Entity Type	Description	Core Set Attributes
MPE-GROUP	Describes an MPE group.	scope-owner scope-changed date-created date-changed sensitivity id-number
NETWORK-CONTROL	Describes the global information for the Network Directory.	scope-owner scope-changed date-created date-changed sensitivity id-number directory-version node-name-changed
NETWORK-DOMAIN	Describes a network domain which is the name space above NODE	scope-owner scope-changed date-created date-changed sensitivity id-number
NETWORK-ORGANI-ZATION	Describes a network organization which is the name space above NODE	scope-owner scope-changed date-created date-changed sensitivity id-number
NODE	Describes a node in a network.	scope-owner scope-changed date-created date-changed sensitivity id-number
RECORD	Describes a record format used in a dataset, table, file or KSAM file.	scope-owner scope-changed date-created date-changed sensitivity id-number byte-length

Table 8: Core Set Entity Types

Entity Type	Description	Core Set Attributes
SYSTEM	Describes an application system	system-type scope-owner scope-changed date-created date-changed sensitivity id-number byte-length
TRANSACTION	Describes a transaction.	scope-owner scope-changed date-created date-changed sensitivity id-number byte-length
USER	Describes a user on a system.	scope-owner scope-changed date-created date-changed sensitivity id-number byte-length

Core Set Relationship Classes

The following relationship classes are included in the core set:

Table 9: Core Set Relationship Classes

Class	Description
accesses	This relationship class classifies those relationship types that define the access the first entity type has to the second entity type.
chains	This relationship class classifies those relationship types where the entity types combine to form a database chain.
contains	This relationship class classifies those relationship types where the first entity type contains the other entity.
key	This relationship class classifies those relationship types where the second entity type is a key for the first entity type.
names	This relationship class classifies those relationship types where the entity types combine to form a qualified name.

Table 9: Core Set Relationship Classes

Class	Description
owns	This relationship class classifies those relationship types where the first entity type owns the other entity.
processes	This relationship class classifies those relationship types where the first entity type processes the second entity type.
redefines	This relationship class classifies those relationship types where the first entity type redefines the second entity type.
references	This relationship class classifies those relationship types where the first entity type references the second entity type.
uses	This relationship class classifies those relationship types where the first entity type uses the second entity type.

Core Set Relationship Types

The following relationship types are included in the core set. Note that in the first column, the entity types are shown in upper case, while the relationship class is shown in lower case. Following this table are diagrams that represent how the relationship types are used to represent various subsystems. A diagram of IMAGE entity types and relationships is included in Chapter 7, Subsystem Support.

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
CATEGORY contains CATEGORY	This relationship type category is contained in a category.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
CATEGORY contains ELEMENT	This relationship type establishes the elementthat is contained in a category.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
COPYLIB key KSAMFILE	This relationship type establishes the KSAM file key for a copylib.	scope-owner date-created date-changed scope-changed relationship-position sensitivity

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
DEVICE-CLASS contains DEVICE	This relationship type establishes the devices contained in a device class	scope-owner date-created date-changed scope-changed relationship-position sensitivity
ELEMENT contains ELEMENT	This relationship type establishes a child element which is a part of the parent element.	scope-owner date-created date-changed scope-changed relationship-position sensitivity back-reference-flag element-type byte-offset display-length decimal blank justify
ELEMENT contains HP-CONDITION-NAME	This relationship type establishes a COBOL condition name that is subordinate to the elemen	scope-owner date-created date-changed scope-changed relationship-position sensitivity
ELEMENT redefines ELEMENT	This relationship type establishes that the first element redefines the second element.	scope-owner date-created date-changed scope-changed relationship-position sensitivity element-type display-length decimal blank justify
ELEMENT references ELEMENT	This relationship type establishes a reference to the second element by the first element, for example, a Pascal typereference	scope-owner date-created date-changed scope-changed relationship-position sensitivity

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
ELEMENT contains IMAGE-CLASS	This relationship type establishes the IMAGE data items secured by the IMAGE security class.	scope-owner date-created date-changed scope-changed relationship-position sensitivity access
FILE contains RECORD	This relationship type establishes a record layout for the file.	scope-owner date-created date-changed scope-changed relationship-position primary-record sensitivity
FILE uses DEVICE	This relationship type establishes that a file will use a device for output	scope-owner date-created date-changed scope-changed relationship-position sensitivity cctl-flag
FILE uses DEVICE-CLASS	This relationship type establishes that a file will use a device class for output	scope-owner date-created date-changed scope-changed relationship-position sensitivity cctl-flag
FORMSFILE contains FORM	This relationship type establishes that a form is a part of a formsfile.	scope-owner date-created date-changed scope-changed relationship-position sensitivity head-form

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
FORM contains ELEMENT	This relationship type establishes that an element is a part of a form.	scope-owner date-created date-changed scope-changed relationship-position sensitivity element-type byte-offset display-length decimal field-enhancement field-type field-number
HPDBE-FILE contains FILE	This relationship type defines the system file name of a dbefile (ie. the MPE file name).	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPDBE-FILESET contains HPDBE-FILE	This relationship type establishes that an HPSQL dbefile is contained in an HPSQL dbefileset.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPDBE-FILESET contains HPSQL-TABLE	The relationship type establishes that the rows of an HPSQL table are contained in an HPSQL dbefileset	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPDBE-LOGFILE contains FILE	The relationship type defines the system file name of a dbefile (ie. the MPE file name).	scope-owner date-created date-changed scope-changed relationship-position sensitivity

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
HPDBENVIRONMENT contains HPSQL-AUTH-NAME	The relationship type establishes that an HPSQL authorization group is contained in an HPSQL DBEnvironment.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPDBENVIRONMENT contains HPDBE-FILESET	The relationship type establishes that an HPSQL dbefileset is contained in an HPSQL DBEnvironment	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPDBENVIRONMENT contains HPDBE-LOGFILE	The relationship type establishes that a logfile is contained in a DBEnvironment	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPDBENVIRONMENT contains HPSQL-TABLE	The relationship type establishes that an HPSQL table is contained in an HPSQL DBEnvironment.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPDBENVIRONMENT contains HPSQL-VIEW	The relationship type establishes that an HPSQL view is contained in an HPSQL DBEnvironment.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPDBENVIRONMENT contains IMAGE-DATABASE	The relationship type establishes that an IMAGE database is contained in a DBEnvironment.	scope-owner date-created date-changed scope-changed relationship-position sensitivity

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
HPSQL-AUTH-NAME contains USER MPE-ACCOUNT	The relationship type establishes an HPSQL authorization name that represents an HPSQL user.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPSQL-AUTH-NAME contains HPSQL-AUTH-NAME	The relationship type establishes the HPSQL authorization groups and HPSQL users that belong to an HPSQL authorization group	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPSQL-AUTH-NAME accesses ELEMENT HPSQL-TABLE	The relationship type establishes that an HPSQL authorization name has update authority to an element within the HPSQL table.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPSQL-AUTH-NAME accesses ELEMENT HPSQL-VIEW	The relationship type establishes that an HPSQL authorization name has update access to an element within the HPSQL view.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPSQL-AUTH-NAME accesses HPSQL-TABLE	The relationship type establishes that an HPSQL authorization name has access to any element within the HPSQL table.	scope-owner date-created date-changed scope-changed relationship-position sensitivity hpsql-alter-auth hpsql-delete-auth hpsql-index-auth hpsql-insert-auth hpsql-select-auth hpsql-update-auth

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
HPSQL-AUTH-NAME accesses HPSQL-VIEW	The relationship type establishes that an HPSQL authorization name has access to any element within the HPSQL view	scope-owner date-created date-changed scope-changed relationship-position sensitivity hpsql-alter-auth hpsql-delete-auth hpsql-index-auth hpsql-insert-auth hpsql-select-auth hpsql-update-auth
HPSQL-AUTH-NAME accesse MODULE	The relationship type establishes that an HPSQL authorization name has the authority to run a module	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPSQL-AUTH-NAME owns HPSQL-AUTH-NAME	The relationship type establishes that an HPSQL authorization name owns an HPSQL authorization group	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPSQL-AUTH-NAME owns HPSQL-TABLE	The relationship type establishes that an HPSQL authorization name owns an HPSQL table.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPSQL-AUTH-NAME owns HPSQL-VIEW	The relationship type establishes that an HPSQL authorization name owns an HPSQL view	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPSQL-AUTH-NAME owns MODULE	The relationship type establishes that an HPSQL authorization name owns a module	scope-owner date-created date-changed scope-changed relationship-position sensitivity

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
HPSQL-INDEX contains ELEMENT	The relationship type establishes the elements that make up an HPSQL index	scope-owner date-created date-changed scope-changed relationship-position sensitivity
HPSQL-TABLE contains RECORD	The relationship type establishes a record layout for an HPSQL table.	scope-owner date-created date-changed scope-changed relationship-position sensitivity primary-record
HPSQL-TABLE key HPSQL-INDEX	The relationship type establishes an index of an HPSQL table.	scope-owner date-created date-changed scope-changed relationship-position sensitivity hpsql-clustering unique
HPSQL-VIEW contains RECORD	The relationship type establishes the record layout of a view.	scope-owner date-created date-changed scope-changed relationship-position sensitivity primary-record
IMAGE-DATASET ELEMENT ELEMENT IMAGE-DATASET IMAGE-DATABASE chains	This five-way relationship type establishes the path between an IMAGE master or relation data set and an IMAGE detail or relation data set, and gives the search and sort items, and the database which this chain is in. The entity types describe the following:IMAGE-DATASET : the detail or relation dataset ELEMENT : the search item of this chain ELEMENT : the sort item of this chain IMAGE-DATASET : the path master or relation dataset IMAGE-DATABASE : the database this chain is in	scope-owner date-created date-changed scope-changed relationship-position sensitivity primary-flag

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
IMAGE-DATABASE contains IMAGE-CLASS	This relationship type establishes the classes in an IMAGE database.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
IMAGE-DATABASE contains IMAGE-DATASET	This relationship type establishes the data sets in an IMAGE database.	scope-owner date-created date-changed scope-changed relationship-position sensitivity blocking-factor capacity
IMAGE-DATASET contains HPDBE-FILE	This relationship type establishes that a dbefile is contained in an HPImage dataset.	scope-owner date-created date-changed scope-changed relationship-position sensitivity access
IMAGE-DATASET contains IMAGE-CLASS	This relationship type establishes the IMAGE data sets secured by the IMAGE security class.	scope-owner date-created date-changed scope-changed relationship-position sensitivity access
IMAGE-DATASET contains RECORD	This relationship type establishes the record layout of an IMAGE data set.	scope-owner date-created date-changed scope-changed relationship-position sensitivity primary-record
IMAGE-DATASET key ELE-MENT	This relationship type establishes the key data item of an IMAGE master data set.	scope-owner date-created date-changed scope-changed relationship-position sensitivity

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
IMAGE-DATASET uses DEVICE CLASS	This relationship type specifies the device class at which a dataset resides.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
INFORM-CLASS contains IMAGE-CLASS	This relationship type establishes the IMAGE security classes contained in an Inform class to describe Inform user security.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
INFORM-CLASS contains INFORM-GROUP	This relationship type establishes the Inform group that is secured to an Inform security class.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
INFORM-GROUP contains INFORM-GROUP	This relationship type establishes that the child Inform group is a part of the parent Inform group.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
INFORM-GROUP contains ELEMENT FILE	This relationship type establishes that the data element from the MPE file is contained in an Inform group.	scope-owner date-created date-changed scope-changed relationship-position sensitivity link-value element-display
INFORM-GROUP contains ELEMENT IMAGE-DATASET IMAGE-DATABASE	This relationship type establishes that the data element from the IMAGE dataset/ database is contained in an Inform group.	scope-owner date-created date-changed scope-changed relationship-position sensitivity link-value element-display

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
INFORM-GROUP contains ELEMENT KSAMFILE	This relationship type establishes that the data element from the KSAM file is contained in an Inform group.	scope-owner date-created date-changed scope-changed relationship-position sensitivity link-value element-display
INFORM-GROUP contains ELEMENT HPSQL-TABLE HPDBENVIRONMENT	This relationship type establishes that the data element from the HPSQL table/ DBEnvironment is contained in an Inform group.	scope-owner date-created date-changed scope-changed relationship-position sensitivity link-value element-display
KSAMFILE contains RECORD	This relationship type establishes a record format for a KSAM file.	scope-owner date-created date-changed scope-changed relationship-position sensitivity primary-record
KSAMFILE key KSAMFILE	This relationship type establishes for a KSAM data file its KSAM key file.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
KSAMFILE key ELEMENT	This relationship type establishes a key for a KSAM file.	scope-owner date-created date-changed scope-changed relationship-position sensitivity primary-flag unique

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
LOCATION contains COPYLIB	This relationship type establishes that a copylib resides at a location.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
LOCATION contains DEVICE	This relationship type establishes that a device resides at a location.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
LOCATION contains DICTIONARY	This relationship type establishes that a dictionary resides at a location.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
LOCATION contains DOCUMENT	This relationship type establishes that a document resides at a location.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
LOCATION contains HPDBENVIRONMENT	The relationship type	scope-owner date-created date-changed scope-changed relationship-position sensitivity
LOCATION contains IMAGE-DATABASE	The relationship type establishes the location of a database.	scope-owner date-created date-changed scope-changed relationship-position sensitivity

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
LOCATION contains FILE	The relationship type establishes that a file is at a location.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
LOCATION contains FORMS-FILE	The relationship type establishes that a VPLUS forms file is at a location.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
LOCATION contains KSAM-FILE	The relationship type establishes that a KSAM file is at a location.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
LOCATION contains MODULE	This relationship type establishes that a module is at a location.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
LOCATION names MPE-GROUP MPE-ACCOUNT	This relationship type names the MPE group and MPE account that make up a location.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
LOCATION names NODE NET-WORK-DOMAIN NETWORK-ORGANIZATION	This relationship type names the node, network domain and network organization that make up a location.	scope-owner date-created date-changed scope-changed relationship-position sensitivity

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
LOCATION contains INFORM-REPORT	This relationship type establishes that a report is at a location.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
MODULE contains MODULE	This relationship type establishes that the child module is a part of the parent module.	scope-owner date-created date-changed scope-changed relationship-position sensitivity directive
MODULE processes ELEMENT	This relationship type establishes that an element is processed by a module.	scope-owner date-created date-changed scope-changed relationship-position sensitivity parm-flag parm-number pass-method
MODULE processes FILE	This relationship type establishes that a file is processed by a module.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
MODULE processes IMAGE-DATABASE	This relationship type establishes that a database is processed by a module.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
MODULE processes KSAMFILE	This relationship type establishes that a KSAM file is processed by a module.	scope-owner date-created date-changed scope-changed relationship-position sensitivity

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
MODULE processes MODULE	This relationship type establishes that the parent module calls the child module.	scope-owner date-created date-changed scope-changed relationship-position sensitivity directive
MPE-ACCOUNT contains USER	This relationship type establishes that a user is assigned to an MPE account.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
MODULE contains TRANSACTION	This relationship type establishes that a transaction is part of a module.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
NODE NETWORK-DOMAIN NETWORK-ORGANIZATION names	This 3-way relationship type names a fully qualified Network Directory node name.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
RECORD contains ELEMENT	This relationship type establishes that an element is part of a record.	scope-owner date-created date-changed scope-changed relationship-position sensitivity back-reference-flag element-type byte-offset display-length decimal blank justify not-null

Table 10: Core Set Relationship Types

Relationship Type	Description	Core Set Attributes
RECORD redefines RECORD	This relationship type establishes that the first record redefines the second record.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
SYSTEM contains SYSTEM	This relationship type establishes that a child system is part of a parent system.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
SYSTEM contains MODULE	This relationship type establishes that a module is part of a system.	scope-owner date-created date-changed scope-changed relationship-position sensitivity
	This relationship type	
	This relationship type	
	This relationship type	

Core Set Domains

The following domains are included in the core set:

Table 11: Core Set Domains

Domain	Description
DICTIONARY-REPORT	This domain stores SDMAIN reports.
INFORM-REPORT	This domain stores RAPID reporting information.
the common domain	This domain is created when you initialize System Dictionary. It is unnamed and you use blanks to access the common domain. (Refer to Chapter 4 for more information.)

Core Set Scopes

The following scopes are included in the core set:

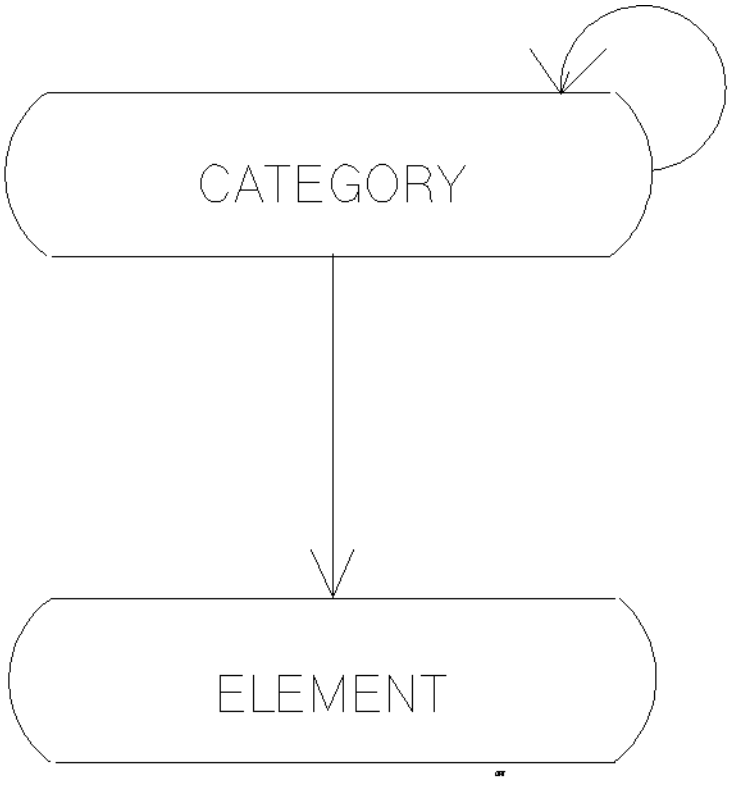
Table 12: Core Set Scopes

Scope	Description
CORESET	This scope owns all core set definitions and structures.
the DA scope	This scope (Dictionary Administrator) has access to the entire dictionary and has all capabilities. The DA scope name is created when you initialize System Dictionary. (Refer to Chapter 5 for more information.)

Core Set Diagrams

The following figures illustrate the core set structures of the following relationships/subsystems:

- **Category** (Figure 6-4)
- **Location** (Figure 6-5)
- **Module** (Figure 6-6)
- **INFORM** (Figure 6-7)
- **KSAM files** (Figure 6-8)
- **MPE files** (Figure 6-9)
- **Formsfile (VPLUS)** (Figure 6-10)



CATEGORY RELATIONSHIP TYPES

Figure 6-4. Category Relationship Types

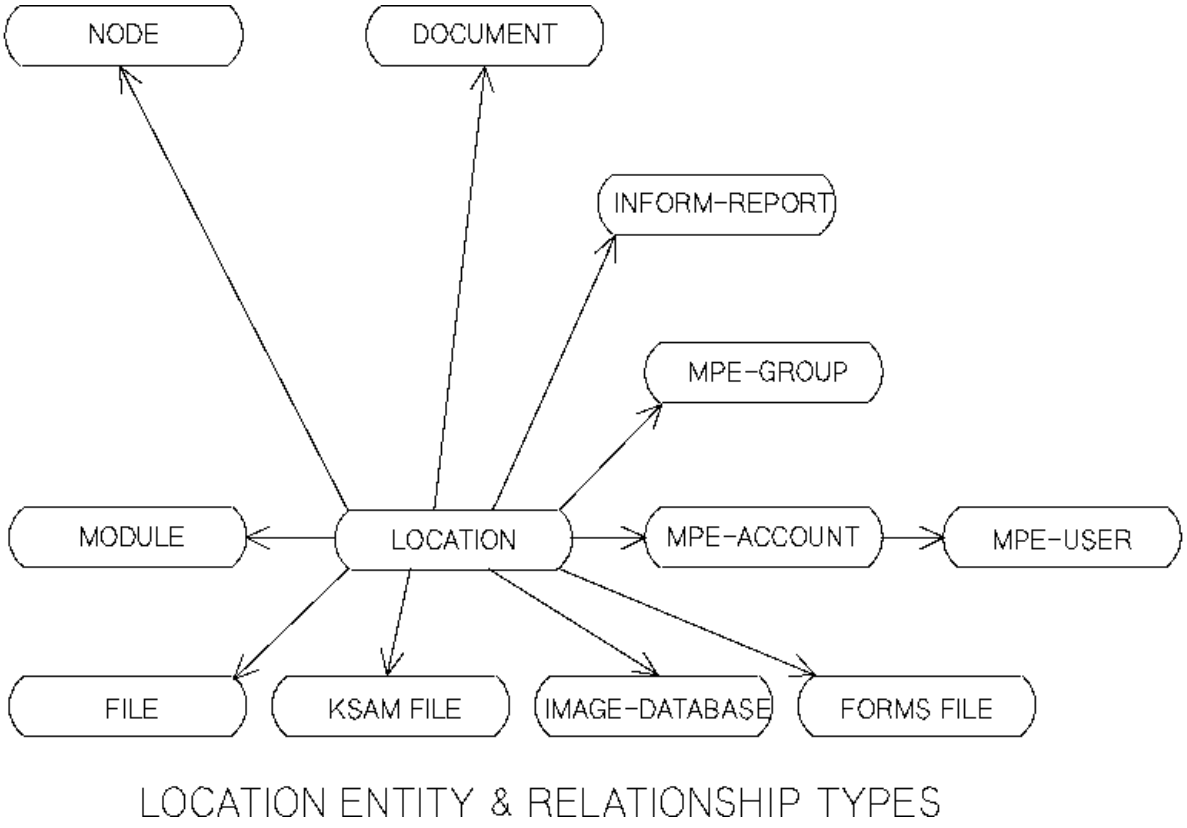


Figure 6-5. Location Entity and Relationship Types

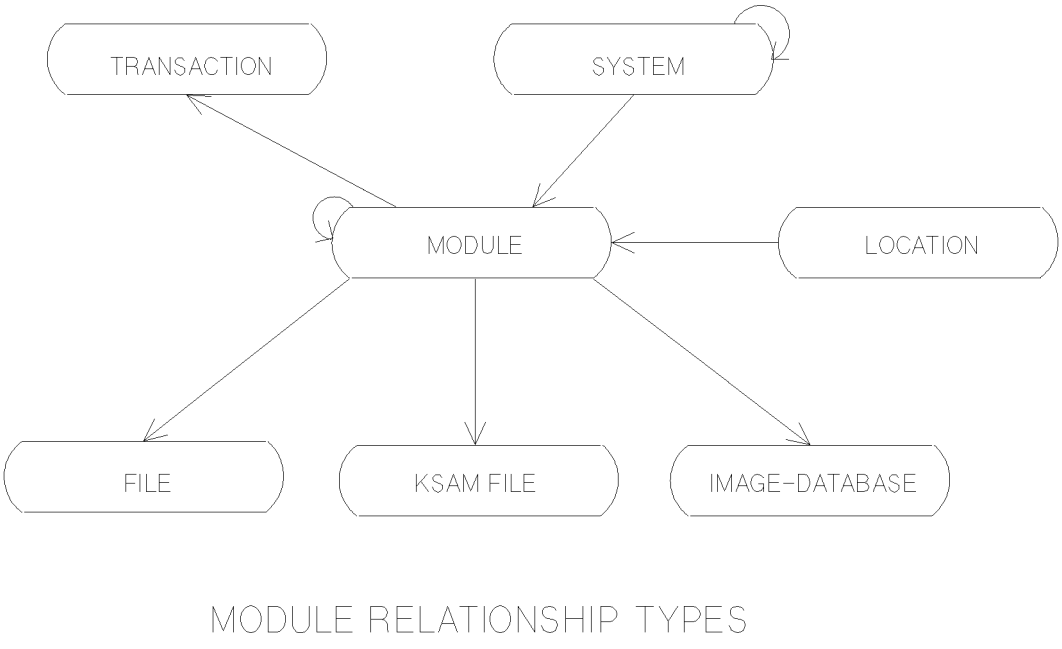


Figure 6-6. Module Relationship Types

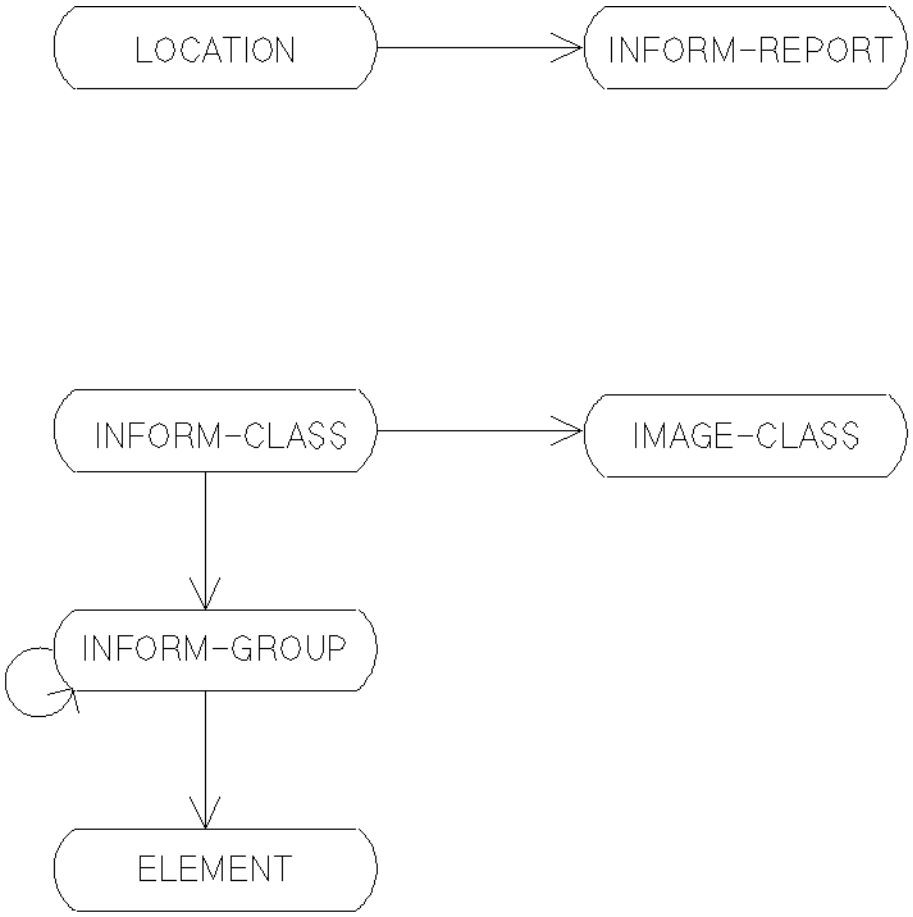
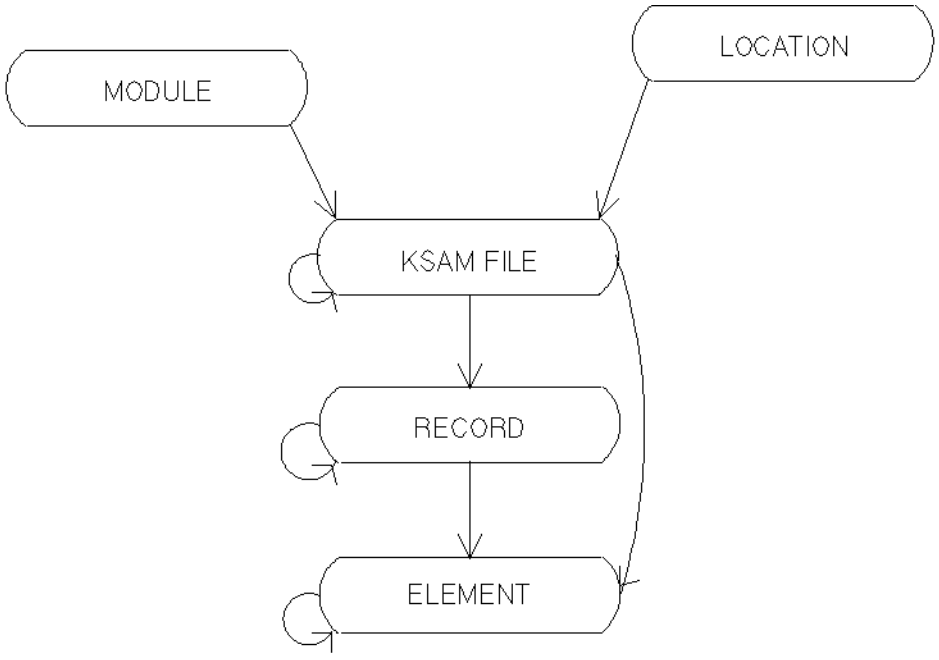


Figure 6-7. Inform Entity and Relationship Types



KSAM FILE SYSTEM ENTITY TYPES & RELATIONSHIP TYPES

KFSETRT

Figure 6-8. KSAM File System Entity & Relationship Types

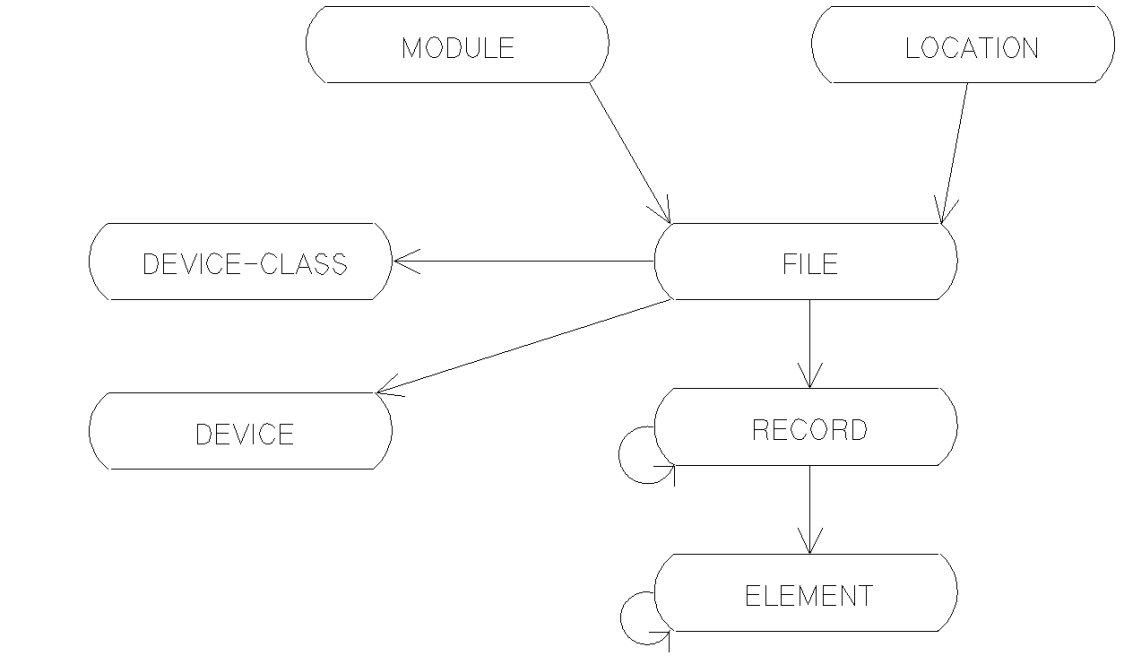


Figure 6-9. MPE File System Entity & Relationship Types

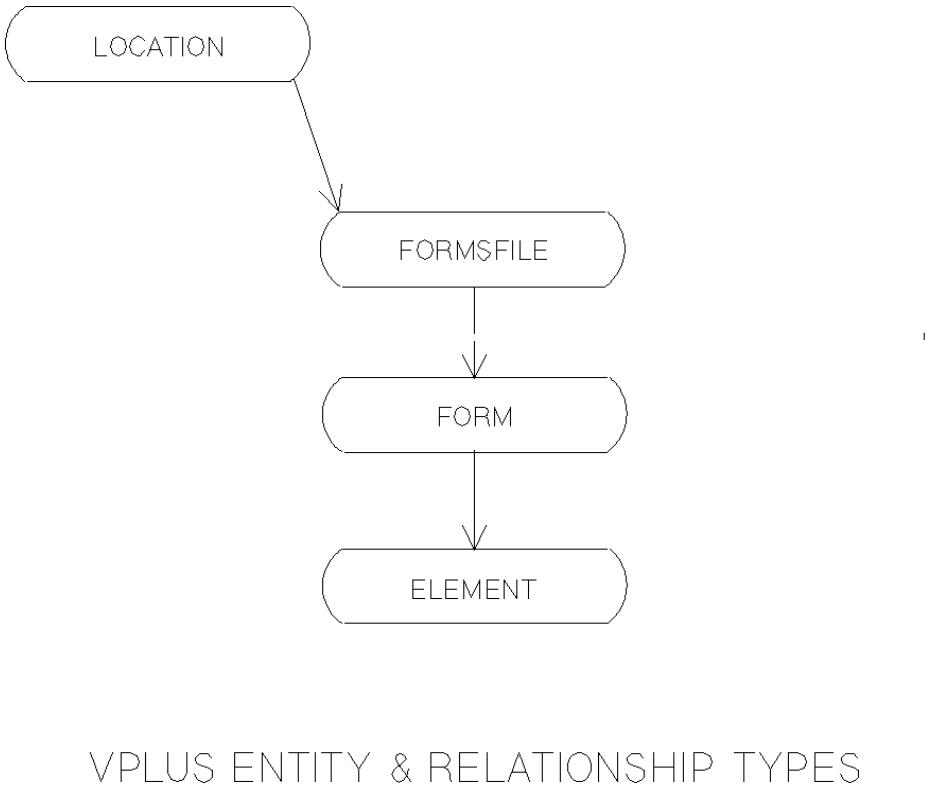


Figure 6-10. VPLUS Entity and Relationship Types

7 Subsystem Support

Overview

This chapter shows how the core set of definitions provided with System Dictionary supports various HP subsystems.

The System Dictionary core set includes support of TurboIMAGE, HP IMAGE and HP SQL database definitions. Examples of a TurboIMAGE database, an HP IMAGE database, and an HP SQL database are provided to show how IMAGE, HP IMAGE, and HP SQL objects are mapped into the dictionary. Each example includes a figure which illustrates how the respective entity types and relationship types for that example are included in the core set.

NOTE In the examples, **bold** print indicates TurboIMAGE, HP IMAGE, or HPDB terminology. *Italic* print indicates System Dictionary terminology.

Mapping TurboIMAGE Into System Dictionary

The TurboIMAGE objects listed in Figure 7-1 are supported in the core set of the System Dictionary. For each of these objects, an example is provided to show how to define each object in the dictionary. All of the examples for these TurboIMAGE objects refer to the database in Figure 7-2.

data item

data set

chain

search item

sort item

path

user class

password

data item class list

data set class list

database

Figure 7-1. TurboIMAGE Objects

```
begin data base STORE;


passwords:
```

```
  1  clrk;   << CLERICAL class >>
  2  bbb12;  << BUYER class    >>
  3  mgr;    << MANAGER class  >>

items:
  ACCOUNT,      J2;
  CITY,         X40 (1,2/3);
  DATE,         X6;
  DELIV-DATE,   X6  (/1);
  NAME,         3X30;
  PURCH-DATE,   X6  (2/1);
  TOTAL,        J2;

 sets:
  name:     CUSTOMER, manual (1/2,3);
  entry:    ACCOUNT(1),
            NAME,
            CITY;
  capacity: 101;

  name:     DATE-MAST,automatic;
  entry:    DATE(2);
  capacity: 101;

  name:     SALES, detail (2/1);
  entry:    ACCOUNT(CUSTOMER(PURCH-DATE)),
            TOTAL,
            PURCH-DATE(!DATE-MAST),
            DELIV-DATE(DATE-MAST);
  capacity: 101;

end.
```

Figure 7-2. STORE TurboIMAGE database schema

Data Item

A data item is an element entity. The sub-item count, type designator and sub-item length of a data item is stored in the element attributes count, element-type and byte-length, respectively.

Example

To define the data item NAME, create the element NAME with the following attributes:

count = 3

element-type = X

byte-length = 30

Data Set

A data set is an image-dataset entity. The data set type is stored in the data set attribute image-dataset-type. A manual master data set has the image-dataset-type MANUAL. An automatic master data set has the image-dataset-type AUTOMATIC. A detail data set has the image-dataset-type DETAIL.

For each data set there should be a record entity occurrence that defines the record layout for that data set entry. Each element (data item) that is in that data set should be related to the record with a specified byte-offset for the relative position of that element within that record. The record should be related to the data set entity occurrence that it describes. This relationship is established with the image-dataset contains record relationship type.

Typically, only one record is related to a data set to define a data set entry layout. However, you have the flexibility of defining more than one record layout for an image-dataset entity. Each image-dataset can have one primary record layout which is indicated with the image-dataset contains record relationship type attribute primary-record.

Example

To define the data set CUSTOMER, create the data set CUSTOMER with image-dataset-type equal to MANUAL.

Create the record CUSTOMER-REC to define the record layout of data set CUSTOMER.

Create the elements ACCOUNT, NAME, and CITY.

Relate each of these elements to record CUSTOMER-REC using the record contains element relationship type. The byte-offset for ACCOUNT is 1, for NAME is 5, for CITY is 95.

Relate CUSTOMER-REC to CUSTOMER using the image-dataset contains record relationship type.

Search Item

The search item of a master data set is defined by the image-dataset key element relationship type. That is, the element (search item) is related to the MANUAL data set for which it is a key. Note that the primary record layout for this data set should contain this element. Also note that there is no need to establish a key for an AUTOMATIC data set since the primary record layout should contain only the search item.

Example

To define the search item for data set CUSTOMER, relate the element ACCOUNT to the data set CUSTOMER with the image-dataset key element relationship type.

Chain

Every chain within a detail data set is fully defined by:

1 The detail data set the chain is in

2 The search item

3 The sort item, if any

4 The path to a master data set

5 The database the chain is in

This is considered a five-way relationship. To define this five-way relationship in the dictionary, use the five-way relationship type:

image-dataset

element

element

image-dataset

image-database

where the relationship class is chains. Note that the order of these entity types is very important.

Information on N-ary (3, 4, 5, and 6-way) relationships is located in Chapter 3 of this manual.

Example

To define the ACCOUNT chain in data set SALES, relate the detail data set SALES, the element ACCOUNT, the element PURCH-DATE, the data set CUSTOMER, the database STORE, in that order, with the relationship class chains. To indicate that this is the primary path, set the attribute primary-flag to true.

User Class Number

A user class number is defined as an image-class entity. Note that in the dictionary the class can be given a name instead of the actual image class number. Password is specified in the image-class attribute password. The user class number is stored in the image-class attribute class-number.

Example

To define the user class number 1, create the image-class CLERICAL (or any name that implies which class this is) with password equal to clrk, and class-number equal to 1. To establish that class-number 1 is in database STORE, use the relationship type image-database contains image-class.

Data Item Class List

A data item class list is defined by the element contains image-class relationship type. An element (data item) should be related to each image-class in its class list with READ/WRITE access specified in the element contains image-class relationship type attribute access.

Example

To define the class list for element DELIV-DATE, relate the element DELIV-DATE to the class CLERICAL using the ELEMENT contains IMAGE-CLASS relationship type with access equal to W.

Data Item Class List

A data set class list is defined by the image-dataset contains image-class relationship type. A data set should be related to each class in its class list with READ/WRITE access specified in the image-dataset contains image-class relationship type attribute access.

Example

To define the class list for data set SALES, relate the data set SALES to the classes CLERICAL and BUYER using the image-dataset contains image-class relationship type with access equal to W for CLERICAL and R for BUYER.

Database

A database is an image-database entity. The image-database contains image-dataset relationship type defines which data sets are contained in an TurboIMAGE data base. Attributes of this relationship type are block and capacity.

Example

To define the TurboIMAGE database STORE, create the image-database STORE. To establish which data sets are in STORE, relate the data sets CUSTOMER, DATE-MAST and SALES to database STORE using the image-database contains image-dataset relationship type. For each of these relationships, the

attribute CAPACITY is set to 101.

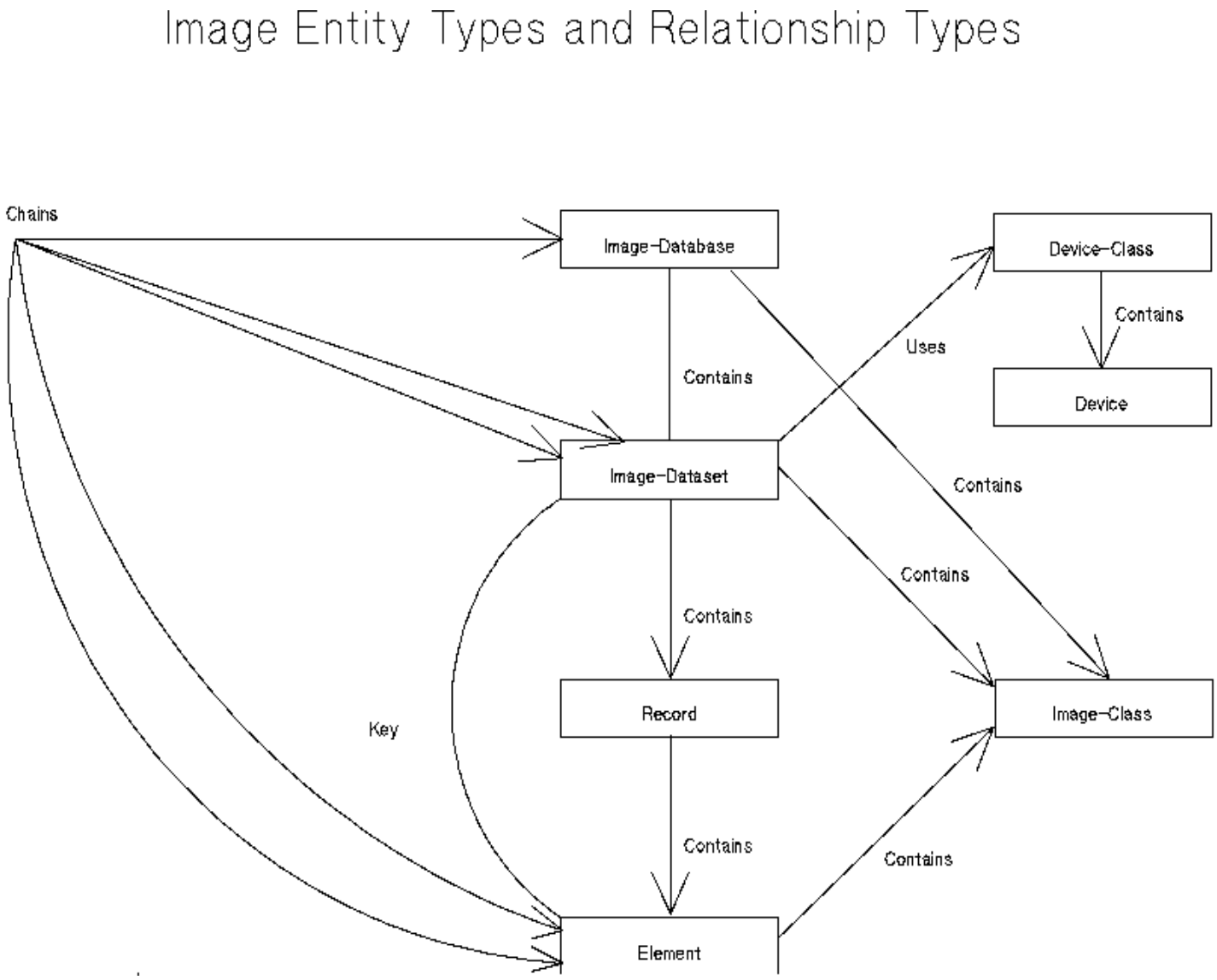


Figure 7-3

Mapping HP IMAGE Into System Dictionary

The System Dictionary core set supports the HP IMAGE objects listed below in Figure 7-4. This subsection provides a description and an example of each of these objects, to show how to define them in the dictionary. All of the examples for these HP IMAGE objects refer to the database in Figure 7-5.

data item

data set

chain

search item

SEARCH ITEM

sort item
path
security class
password
data item class list
data set class list
database
DBEnvironment
DBEFile

Figure 7-4. HP IMAGE Objects

```
begin database STORE;

passwords:

 1  clrk;   << CLERICAL class >>
 2  bbb12;  << BUYER class    >>
 3  mgr;    << MANAGER class  >>

items:
 ACCOUNT,     J2;
 CITY,        X40 (1,2/3);
 DATE,        X6;
 DELIV-DATE,  X6  (/1);
 NAME,        3X30;
 PURCH-DATE,  X6  (2/1);
 TOTAL,       J2;

sets:

 name:    CUSTOMER, manual (1/2,3);
 entry:   ACCOUNT(1),
          NAME,
          CITY;
```

```
capacity: 101;

name:    DATE-MAST,automatic;
entry:   DATE(2);
capacity: 101;

name:    SALES, detail (2/1);
entry:   ACCOUNT(CUSTOMER(PURCH-DATE,DELIV-DATE)),
         TOTAL,
         PURCH-DATE(!DATE-MAST),
         DELIV-DATE(DATE-MAST);
capacity: 101;

end.
```

Figure 7-5. STORE HP IMAGE Database Schema

Data Item

A data item is an ELEMENT entity. The sub-item count, type designator and sub-item length of a data item is stored in the ELEMENT attributes count, element-type and byte-length, respectively.

Example

To define the data item NAME, create the ELEMENT NAME with the following attributes:

```
count        = 3
element-type  = X
byte-length   = 30
```

Data Set

A data set is a IMAGE-DATASET entity. The data set type is stored in the data set attribute image-dataset-type. A manual master data set has the image-dataset-type MANUAL. An automatic master data set has the image-dataset-type AUTOMATIC. A relation data set has the image-dataset-type RELATION. A detail data set has the image-dataset-type DETAIL.

For each data set there should be a RECORD entity occurrence that defines the record layout for that data set entry. Each ELEMENT (data item) that is in that data set should be related to the RECORD with a specified byte offset for the relative position of that ELEMENT within that RECORD. The RECORD should be related to the MANUAL, AUTOMATIC, RELATION, or DETAIL data set entity occurrence that it describes. This relationship is established with the IMAGE-DATASET contains RECORD relationship type. Typically, only one record is related to a data set to define a data set entry layout. However, you have the flexibility of defining more than one record layout for an IMAGE-DATASET entity. Each IMAGE-DATASET can have one primary record layout which is indicated with the IMAGE-DATASET contains RECORD relationship type attribute primary-record. To define the DBEFiles associated to a data set, use

the IMAGE-DATASET contains HPDBE-FILE relationship type.

Example

To define the data set CUSTOMER, create the data set CUSTOMER with image-dataset-type equal to MANUAL.

Create the RECORD CUSTOMER-REC to define the record layout of data set CUSTOMER.

Create each of the ELEMENTs ACCOUNT, NAME and CITY.

Relate each of these ELEMENTs to RECORD CUSTOMER-REC using the RECORD contains ELEMENT relationship type. The byte-offset for ACCOUNT is 1, for NAME is 5, for CITY is 95.

Relate CUSTOMER-REC to CUSTOMER using the IMAGE-DATASET contains RECORD relationship type.

Search Item

The search item of a master or relation data set is defined by the IMAGE-DATASET key ELEMENT relationship type. That is, the ELEMENT (search item) is related to the MANUAL or RELATION data set for which it is a key. Note that the primary record layout for this data set should contain this ELEMENT. Also note that there is no need to establish a key for an AUTOMATIC data set since the primary record layout should contain only the search item.

Example

To define the search item for data set CUSTOMER, relate the ELEMENT ACCOUNT to the data set CUSTOMER with the IMAGE-DATASET key ELEMENT relationship type.

Chain

Every chain within a detail data set is fully defined by:

1. The child (detail or relation) data set the chain is in
2. The search item
3. The sort item, if any
4. The path to a parent (master or relation) data set
5. The database the chain is in

This is considered a five-way relationship. To define this five-way relationship in the dictionary, use the five-way relationship type:

image-dataset

element

element

image-dataset

image-database

where the relationship class is chains. Note that the order of these entity types is very important. Information regarding the order of entity types in a relationship type is located in Chapter 3 of this manual.

Example

To define the ACCOUNT chain in data set SALES, relate the detail data set SALES, the ELEMENT ACCOUNT, the ELEMENT PURCH-DATE, the data set CUSTOMER, the database STORE, in that order, with the relationship class chains. To indicate that this is the primary path, set the attribute primary-flag to true.

Note that there is more than one sort item for this path. To define the second sort item, create the exact same relationship just described except for the sort item, as follows: Relate the detail data set SALES, the ELEMENT ACCOUNT, the ELEMENT DELIV-DATE, the data set CUSTOMER, the database STORE, in that order. The attribute relationship-position keeps track of the order of the sort items.

Security Class Number

A security class number is defined as an IMAGE-CLASS entity. Note that in the dictionary you must give the class a name and you should reference the class by that name, not the Image class number. Password is an IMAGE-CLASS attribute.

Example

To define the security class number 1, create the IMAGE-CLASS CLERICAL (or any name that implies which class this is) with password equal to clrk, and class-number equal to 1. To establish that class-number 1 is in database STORE, use the relationship type IMAGE-DATABASE contains IMAGE-CLASS.

Data Item Class List

A data item class list is defined by the ELEMENT contains IMAGE-CLASS relationship type. An ELEMENT (data item) should be related to each IMAGE-CLASS in its class list with READ/WRITE access specified in the ELEMENT contains IMAGE-CLASS relationship type attribute access.

Example

To define the class list for ELEMENT DELIV-DATE, relate the ELEMENT DELIV-DATE to the class CLERICAL with the ELEMENT contains IMAGE-CLASS relationship type with access equal to WRITE.

Data Set Class List

A data set class list is defined by the IMAGE-DATASET contains IMAGE-CLASS relationship type. A data set should be related to each class in its class list with READ/WRITE access specified in the IMAGE-DATASET contains IMAGE-CLASS relationship type attribute access.

Example

To define the class list for data set SALES, relate the data set SALES to the classes CLERICAL and BUYER using the IMAGE-DATASET contains IMAGE-CLASS relationship type with access equal to WRITE for CLERICAL and READ for BUYER.

Database

A database is a IMAGE-DATABASE entity. The IMAGE-DATABASE contains IMAGE-DATASET relationship type defines which datasets are contained in an IMAGE database. Attributes of this relationship type are BLOCK and CAPACITY.

Example

To define the Image database STORE, create the IMAGE-DATABASE STORE.

To establish which data sets are in STORE, relate the IMAGE-DATASETs CUSTOMER, DATE-MAST and SALES to IMAGE-DATABASE STORE using the IMAGE-DATABASE contains IMAGE-DATASET relationship type. For each of these relationships, set the attribute CAPACITY to 101.

DBEnvironment

A DBEnvironment is a HPDBENVIRONMENT entity. The HPDBENVIRONMENT contains IMAGE-DATABASE relationship type defines which databases are contained in an DBEnvironment. The HPDBENVIRONMENT contains HPDBE-LOGFILE relationship type defines which log files are

contained in a DBEnvironment.

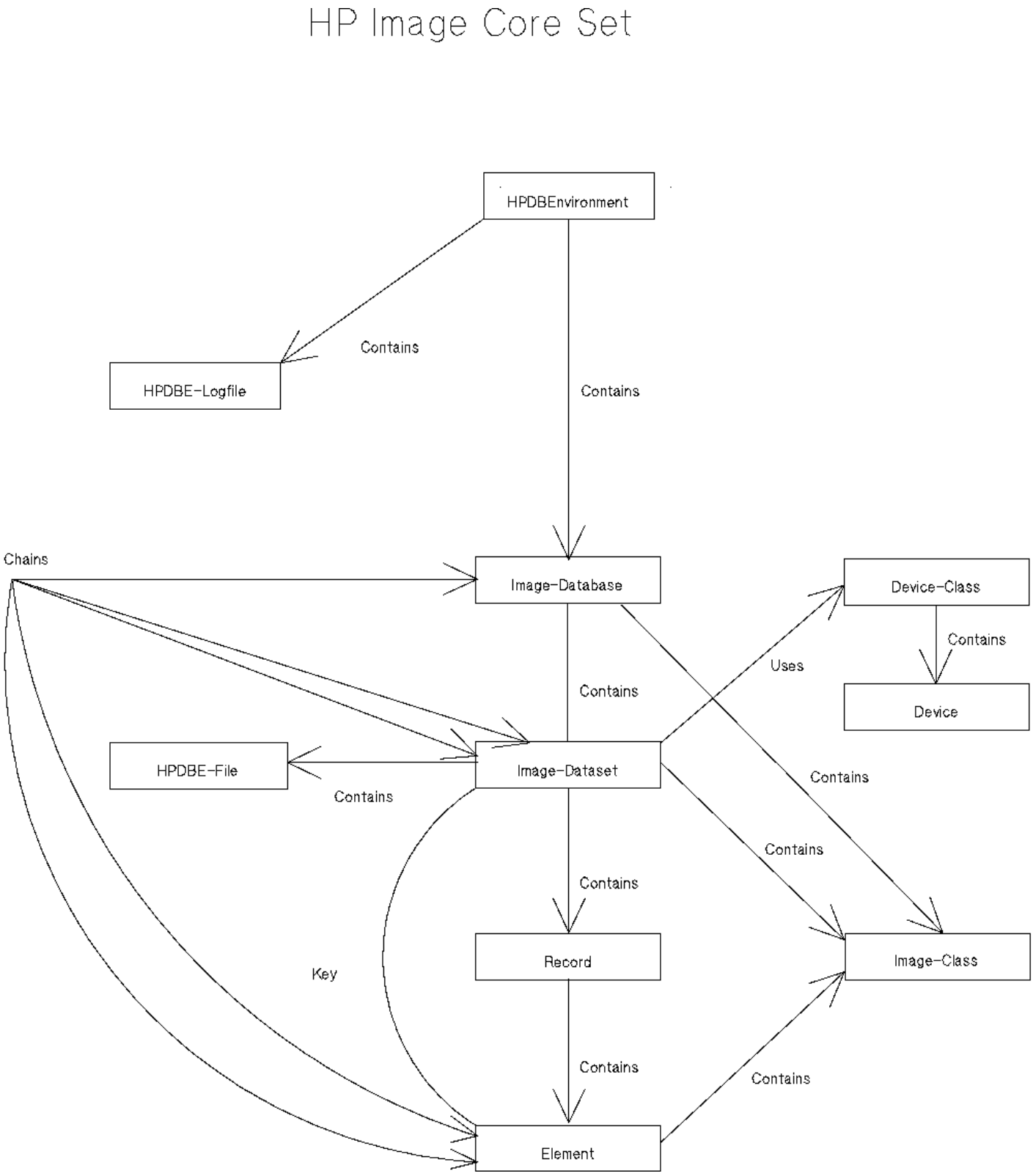


Figure 7-6

Mapping HP SQL Into System Dictionary

The System Dictionary core set supports the HP SQL objects listed below in Figure 7-7. This subsection provides a description of each of these objects, to show how to define them in the dictionary.

column
table
view
index
DBEFileSet
DBEFile
module
authorization group
user
owner
grants
DBEnvironment

Figure 7-7. HP SQL Objects

Column

A column is an ELEMENT entity. The data type of a column is stored in the element attributes element-type, byte-length, display-length, and decimal. The HP SQL data types map into System Dictionary core set data types as follows:

HP SQL	**System Dictionary**			
element-type	element-type	display-length	byte-length	decimal
CHAR(n)	X	n	n	-
VARCHAR(n)	S	n	*note1	-
INTEGER	I	10	4	-
SHORTINT	I	5	2	-
FLOAT	E	27	8	-
DECIMAL(p[,s])	p	p	*note2	s

*note1: Pascal string byte length

*note2:

if p is odd: byte-length = (p+1)/2

if p is even: byte-length = (p+2)/2

Table

A table is an HPSQL-TABLE entity. The lock mode of a table is defined in the hpsql-table attribute hpsql-

lock-mode.

For each table there should be a RECORD entity occurrence that defines the record layout of that table. The record layout consists of the columns (elements) of that table, which is defined with the RECORD contains ELEMENT relationship type. You should specify a byte offset for the relative position of that element within that record. The record should be related to the HPSQL-TABLE entity occurrence that it describes. This relationship is established with the HPSQL-TABLE contains RECORD relationship type. Typically, only one record is related to a table to define a table record layout. However, you have the flexibility of defining more than one record layout for a table entity. Each table can have one primary record layout which is indicated with the hpsql-table contains record relationship type attribute primary-record. The NOT NULL specification for a column is defined in the attribute not-null of the record contains element relationship type, where the record is the primary record of the table.

The HPDBE-FILESET contains HPSQL-TABLE relationship type defines where the rows of the table are stored.

View

A view is an HPSQL-VIEW entity. The variable length attribute hpsql-select-command defines the select command from which the view is derived.

For each view there should be a RECORD entity occurrence that defines the record layout of that view. The record layout consists of the columns (elements) of that view, which is defined with the RECORD contains ELEMENT relationship type. You should specify a byte offset for the relative position of that element within that record. The record should be related to the hpsql-view entity occurrence that it describes. This relationship is established with the HPSQL-VIEW contains RECORD relationship type. Typically, only one record is related to a view to define a view record layout. However, you have the flexibility of defining more than one record layout for a view entity. Each view can have one primary record layout which is indicated with the HPSQL-VIEW contains RECORD relationship attribute primary-record.

Index

An index is an HPSQL-INDEX entity. To define the columns of the index, use the HPSQL-INDEX contains ELEMENT relationship type. That is, relate each element (column), in the order they appear in the index, to the index entity occurrence. Relate the index occurrence to the table that it is an index of with the HPSQL-TABLE key HPSQL-INDEX relationship type. Attributes of the hpsql-table key hpsql-index relationship type include hpsql-clustering to indicate whether clustering is specified for the index and unique to specify if index values must be unique.

DBEfileset and DBEfile

A DBEFileSet is an HPDBE-FILESET entity. A DBEFile is an HPDBE-FILE entity with attributes hpdbe-file-type to define the DBEFile type and hpdbe-pages to define the number of pages in the DBEFile. The relationship type HPDBE-FILESET contains HPDBE-FILE defines the DBEFiles that belong to a DBEFileSet. The relationship type HPDBE-FILE contains FILE defines the system file name by which the DBEFile is known to MPE.

Module

A module is a module entity.

Authorization Groups and Users

An Authorization Group and an HP SQL user are an HPSQL-AUTH-NAME entity with the attribute

hpsql-auth-name-type to indicate authorization group or user, and with the attributes hpsql-connect-auth, hpsql-dba-auth and hpsql-resource-auth to define the HP SQL special authorities granted an authorization group or user (an HP SQL GRANT). To fully define a 'user' HPSQL-AUTH-NAME entity use the relationship type HPSQL-AUTH-NAME contains USER MPE-ACCOUNT (3-way) to define the user and account that the 'user' is made up of.

The relationship HPSQL-AUTH-NAME contains HPSQL-AUTH-NAME defines the authorization groups and users that belong to an authorization group.

Owner

HP SQL tables, views, modules and authorization groups are owned by either an authorization group or a user. Relationship HPSQL-AUTH-NAME owns HPSQL-TABLE defines the owner of an HP SQL table. Relationship HPSQL-AUTH-NAME owns HPSQL-VIEW defines the owner of an HP SQL view. Relationship HPSQL-AUTH-NAME owns MODULE defines the owner of a module. Relationship HPSQL-AUTH-NAME owns HPSQL-AUTH-NAME defines the owner of an HP SQL authorization group.

Grants

Privileges can be granted to a user or an authorization group regarding access to tables, views and modules.

Relationship type HPSQL-AUTH-NAME accesses HPSQL-TABLE establishes that the user or authorization group has access to an HP SQL table. Relationship type HPSQL-AUTH-NAME accesses HPSQL-VIEW establishes that the user or authorization group has access to an HP SQL view. Both of these relationship types have the attributes hpsql-alter-auth, hpsql-delete-auth, hpsql-index-auth, hpsql-insert-auth, hpsql-select-auth and hpsql-update-auth to define the authorities the user or authorization group has to the table or view.

Relationship type HPSQL-AUTH-NAME accesses MODULE establishes that the user or authorization group has the authority to run the module.

The authority to update a column within a table or view can be granted to a user or an authorization group. Relationship type HPSQL-AUTH-NAME accesses ELEMENT HPSQL-TABLE (3-way) establishes that a user or authorization group has update authority on a column within the specified table. Relationship type HPSQL-AUTH-NAME accesses ELEMENT HPSQL-VIEW (3-way) establishes that a user or authorization group has update authority on a column within the specified view.

HP SQL has a special category of user called PUBLIC. To grant privileges to the PUBLIC, create an hpsql-auth-name called PUBLIC and create any of the 'accesses' relationships listed above that are needed to define what authorities the PUBLIC has been granted. For example, the authorities granted the PUBLIC to table TABLE-ONE are defined by the relationship hpsql-auth-name accesses hpsql-table where the hpsql-auth-name entity is PUBLIC and the hpsql-table entity is TABLE-ONE with the HP SQL authorities attributes set to define which authorities the PUBLIC has to TABLE-ONE.

DBEnvironment

An HP SQL DBEnvironment is an HPDBENVIRONMENT entity. The HPDBENVIRONMENT contains HPSQL-TABLE relationship type defines the tables in the HP SQL DBEnvironment. The HPDBENVIRONMENT contains HPSQL-VIEW relationship type defines the views in the HP SQL DBEnvironment. The HPDBENVIRONMENT contains HPSQL-AUTH-NAME relationship type defines the authorization groups in the HP SQL DBEnvironment. The HPDBENVIRONMENT contains HPDBE-FILESET relationship type defines the DBEFileSets in the HP SQL DBEnvironment. The HPDBENVIRONMENT contains HPDBE-LOGFILE relationship type defines the logfiles used in the HP SQL DBEnvironment. The HPDBE-LOGFILE contains FILE relationship type defines the system file by

which the logfile is known by MPE.

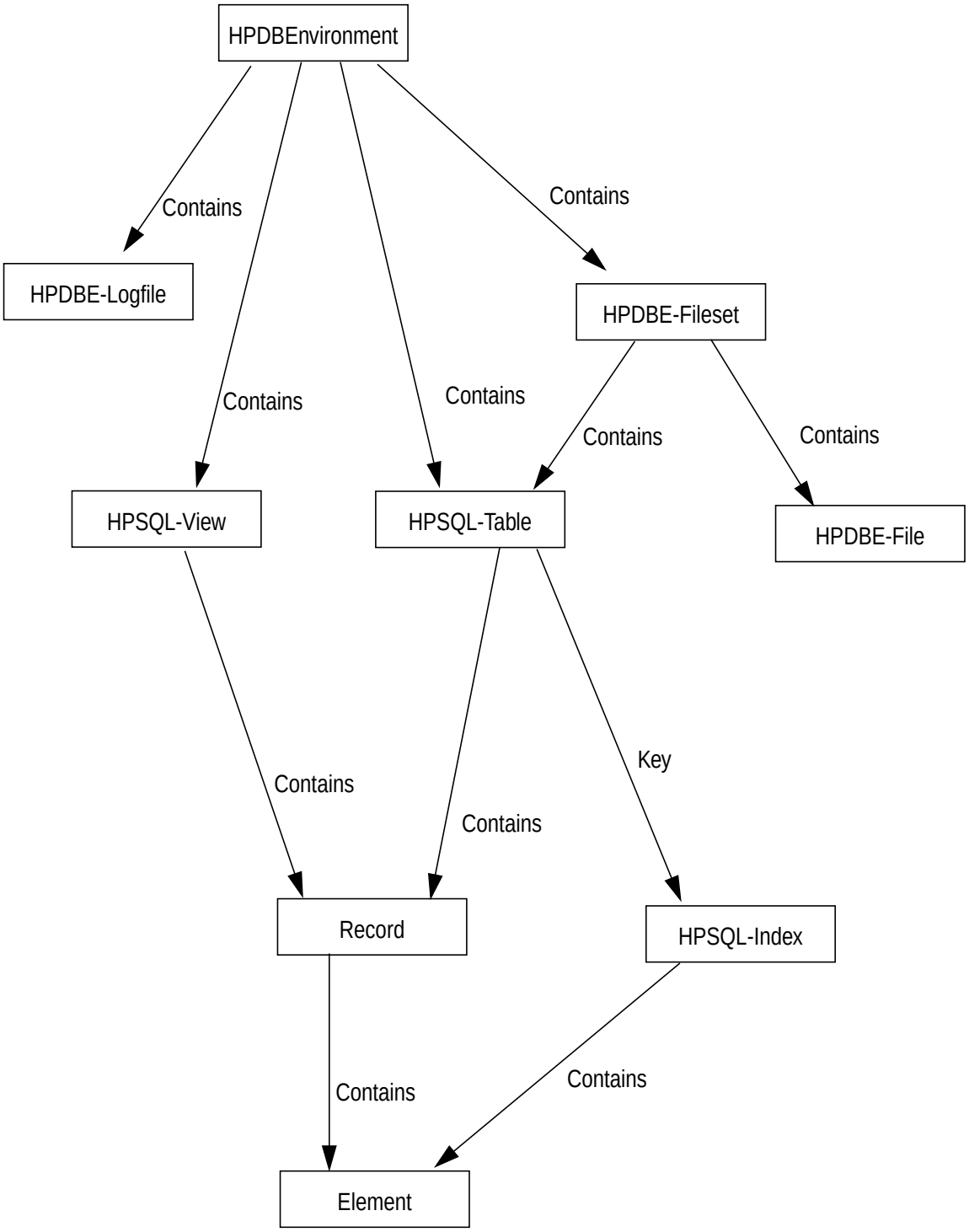


Figure 7-8 HPSQL Coreset

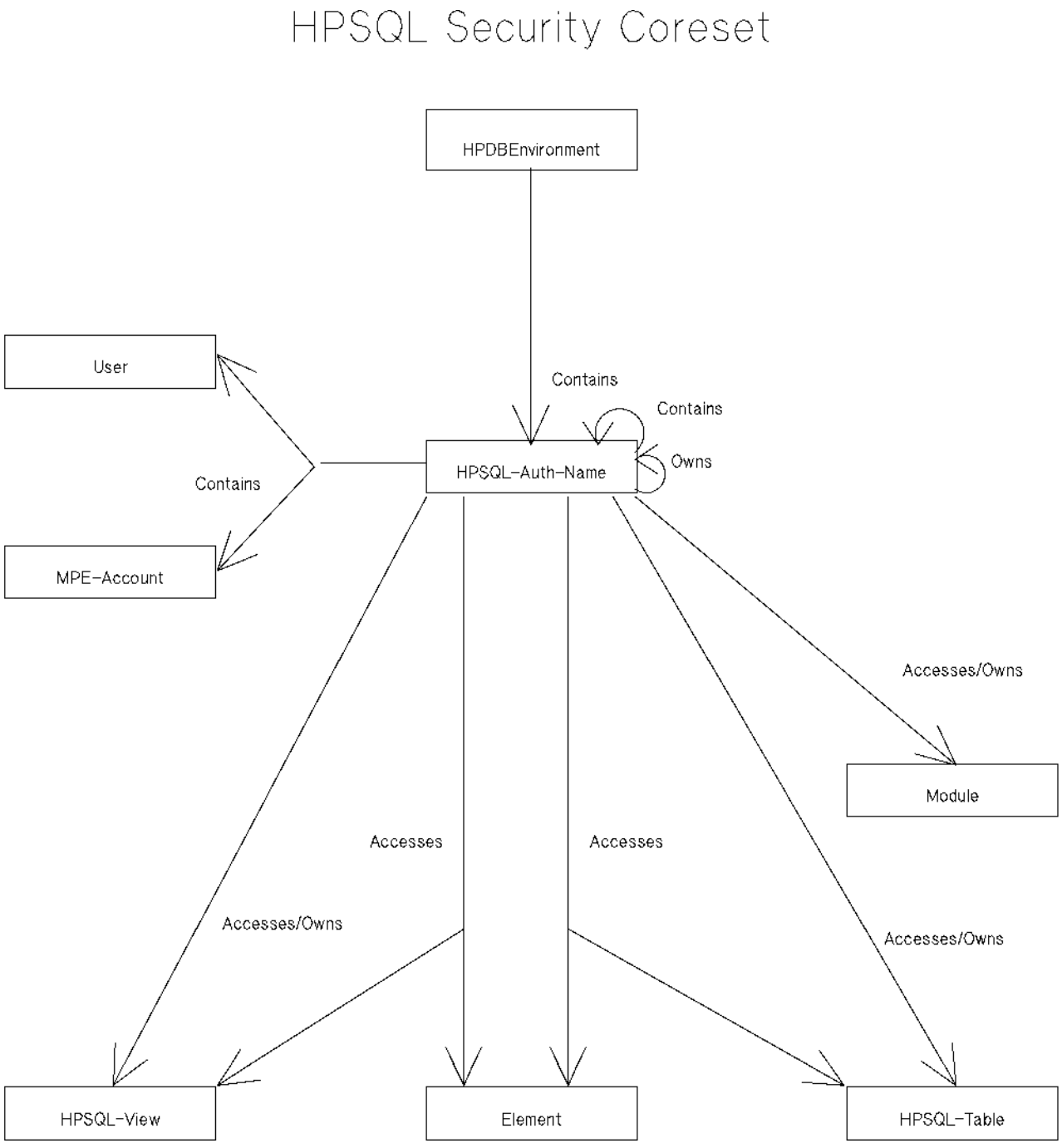

HPSQL Security Coreset
HPDBEnvironment
User
Contains
Contains
Owns
Contains
HPSQL-Auth-Name
MPE-Account
Accesses/Owns
Module
Accesses
Accesses
Accesses/Owns
Accesses/Owns
HPSQL-View
Element
HPSQL-Table

Figure 7-9

A Glossary

This appendix provides a glossary of System Dictionary terms.

Access - The right to read or manipulate a dictionary domain or occurrence.

Access Rights - The rights of a scope to read or manipulate a domain or occurrence, as determined by whether that scope is the owner of the item, or is just associated with it.

Alias Name - Different names associated with different external subsystem uses of an occurrence, including a difference in programming syntax (e.g. the use of the underscore (_) instead of the hyphen (-) in names).

Association - An explicit access assigned between a scope and a domain, entity, or relationship, which has been granted to that scope by the owner scope of the domain, entity, or relationship.

Attribute - An object in the dictionary structure that is a piece of information describing an entity type or relationship type.

Attribute Edit - A value or range of values used for determining if input attribute values are valid when creating or modifying an occurrence. Also used to specify the default attribute value to use when an occurrence is created.

Attribute Prompting - A facility that prompts for attribute values whenever you issue a CREATE, MODIFY, or REPORT command without the ATTRIBUTE-LIST parameter. (SDMAIN only)

Attribute Value - The specific information (e.g. text, numbers, etc) assigned to an attribute, describing a particular *occurrence* of an entity type or relationship type.

Binary Relationship - A relationship involving two entities.

Child Entity - The second entity in a relationship.

Command - The SDMAIN-defined name that specifies the action to be taken.

Common Domain - The primary name space for dictionary occurrences. It is provided with System Dictionary, is represented by a blank name, is owned by the core set, has a sensitivity of *Public*, and can never be modified or deleted.

Compiled Dictionary - A read-only dictionary that contains metadata extracted from the master dictionary. A compiled dictionary consists of one or more flat files.

Core Set - A predefined set of entity types, relationship types, relationship classes, attributes, and domains that are provided with System Dictionary. It also includes the scope CORESET, which owns everything in the core set. A second scope, the Dictionary Administrator scope, is included in the core set, and is also owned by the scope CORESET.

DA scope - See Dictionary Administrator Scope.

DCB - See Dictionary Control Block.

Dictionary Administrator Scope - A special scope provided with the System Dictionary core set, which has unlimited access to all items in the dictionary, and ultimate authority.

Dictionary Control Block - An array of data which contains information about the current status of the dictionary to an intrinsic.

Dictionary Environment - The dictionary environment includes the name of the dictionary that is open,

the scope, the open mode, the name mode, the domain, version, and version status that are used for creating and retrieving definitions.

Domain - A name space within the dictionary. See Common Domain and Local Domain.

E-R Model - See Entity-Relationship model.

Entity - An entity is a description of an object in the information network, and belongs to a specific Entity Type.

Entity List - The ordered list of entities that make up a relationship.

Entity Type - An object in the dictionary structure that classifies entity occurrences. Each entity type is further defined by an associated set of attributes.

Entity-Relationship Model - A logical structure that is general enough that it can describe most, if not all, of the information processing done on a computer network. The entity-relationship model is composed of entity types, relationship types, relationship classes and attributes.

Environment - The computer system hardware and software required for the operation of System Dictionary.

Extended Set - The user-created set of structure definitions within the dictionary; an extension of the Core Set.

External Name - One of two names (see also Internal Name) assigned to every item in the dictionary. It is a customizable and localizable reference that is intended for dictionary end users.

Homonym - The same name used for conceptually different entity occurrences of the same entity type.

Internal Name - One of two names (see also External Name) assigned to every item in the dictionary. An internal name is not changeable, and is intended for use by software products used with System Dictionary which rely on specific names for identification purposes.

Internal Number - An identification number automatically assigned to all dictionary components when they are created. These numbers may be read from the Status array (parameter) of intrinsics used for creation and retrieval of dictionary components and, when used, can greatly increase the efficiency and speed of some dictionary operations.

Keyword-Clause - A keyword clause can be either a single keyword or a keyword followed by an equal sign (=) that is followed by either nothing, a single value, or a list of values separated by commas. The keywords are SDMAIN-defined, while their values are either SDMAIN defined or user-defined. Keyword clauses are separated by semicolons.

List Terminator - A semicolon (**;**) that indicates to the intrinsic using a specific list, that there are no more entries in the list.

Local Domain - A user-created name space that separates a set of names, which includes names used for a different purpose. See also Common Domain.

Locking - A process that allows only one user at a time to access the dictionary. System Dictionary provides two types of locking: automatic, which protects individual operations, and manual, which can protect a sequence of operations.

Logging - A process that can automatically create a log of all dictionary transactions, providing a means to repeat those transactions in the event of data loss.

Macro - A user-defined set of commands that you can save in a file and call using macro names. When you call the macro, each defined command is executed in the same way that it would have been had you entered each command individually. (SDMAIN only)

Master Dictionary - A dictionary that consists of a database and multiple files. A master dictionary can be accessed by any of the System Dictionary intrinsics and commands.

Merge - A process that combines structure, security, and occurrence date of one dictionary into the same or other dictionaries.

Metadata - Descriptive information about data, but not the data itself. Example: a file card in a library, which contains information about a book, but is not the book itself; an address of a building, which provides information about its location, but is not the location itself.

N-ary Relationship - A relationship that involves **N** entities, where **3<=N<=6** (see also Binary Relationship).

Name mode - A parameter set while opening the dictionary, used to cause intrinsics to reference either internal or external names when accessing dictionary items.

Name set - A group of names within the dictionary that includes names for any one of the following types of dictionary definitions: domains, versions in the same domain, entity types, relationship classes, attributes, scopes, and entity occurrences of a specific type that are located in the same domain.

Object-Clause - The user-defined name of the object. This is the specific target of the action specified by the command.

Occurrence - A specific instance of an entity or relationship.

Open mode - One of five dictionary operating modes, set when opening the dictionary for use.

Owner scope - A scope that is directly associated with an object in the dictionary and has all rights to it, because the scope has either created that object, or has been given ownership by the scope that created it or previously owned it.

Parent Entity - The first entity in a relationship. **Password -** A combination of up to 32 special or alphanumeric characters, and/or blanks used for user identification purposes to limit access to data or objects within the dictionary.

Preview - A process that allows the potential results of a merge operation to be seen before the actual merge operation is performed.

Primary Name - The principal name of an entity, not a synonym, that is initially assigned when the entity is created. Whenever an entity name is returned by System Dictionary, the primary name is returned.

Relationship - A logically connected, ordered series of two to six entities, which belongs to a specific Relationship Type.

Relationship class - The specific class of association or logical connection between the entities in a relationship.

Relationship position - The logical order of a child entity (the second entity in a relationship) relative to all other child entities for the same parent entity (the first entity in a relationship) of the same relationship type.

Relationship type - An object in the dictionary structure that classifies relationship occurrences; a logical connection between entity types specified by a series of two to six ordered entity types and a relationship class. Each relationship type is further defined by an associated set of attributes.

Restructuring - A process similar to compiling. Restructuring incorporates all changes made to the dictionary structure in a single session into the working dictionary, and reformats any dictionary occurrences that are affected by those structure changes.

Scope - A security definition within the dictionary environment that sets the level of access a user has to

all objects in the dictionary. It includes up to six scope rights.

Scope Right - One of six specific capabilities associated with a scope. The scope right specifies which dictionary components that scope is allowed to manipulate.

Security - A protection scheme within System Dictionary that limits access to objects in the dictionary to authorized users. The primary elements of dictionary security are scopes In addition, dictionary domains and occurrences each have a sensitivity, which further define their access by a specific scope.

Sensitivity - An access right associated with a dictionary domain or occurrence.

Special Attributes - The set of attributes that are automatically assigned to entity types and relationship types when the types are created.

Status - Information about the success or failure of an intrinsic call. The status is returned as the final parameter of intrinsic calls.

Structure - The part of System Dictionary that includes both core set and extended set entity types, relationship types, relationship classes, and attributes.

Subcommand - The SDMAIN-defined name hat specifies the general target of the action.

Synonym - An alternate name for an entity in the dictionary. A synonym must uniquely identify a given entity.

Variable Length Attribute - An attribute whose value must be explicitly defined, and whose length is dependent upon that value. Example: an attribute *description*, whose value is sixty bytes of text. Therefore, the length of the attribute is sixty.

Version - A set of occurrences within a domain, set apart from other sets within the domain.