CA 2E

Defining a Data Model

Release 8.7

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Documentation Changes

The following documentation updates have been made since the last release of this documentation.

- IBM Limitation - File name is valid system name
  - Change Enhance SQL Naming (see page 195)

Contents

Chapter 1: Introducing Data Modeling

This chapter provides an overview of basic concepts of data modeling and how CA 2E (formerly known as Advantage 2E) handles data modeling. Its purpose is to help you understand data modeling and to prepare you for building and maintaining a data model with CA 2E.

This section contains the following topics:

Understanding Data Modeling

Data modeling is a method of representing the real world. To CA 2E, data modeling is particularly important because a data model is the basis of everything that is designed within CA 2E.

What is a Data Model?

A data model is a structured description of a set of data and its relationships, which represent the business of an organization. A data model bears the same relation to the organization it models as a map does to the terrain it represents.

A data model does not comprise in itself the true structure of an organization's business, but rather an acceptably simplified view of it. This structure may be subjected to certain stringent tests to verify that it can be considered an efficient, self-contained system.

Your data model should be comprehensive and consistent in order to accurately reflect the data and data interconnections of the organization your application supports.

A properly defined and structured data model helps you design a correct database, which is essential to the successful implementation of your application system.

The Life Cycle of Application Development

The life cycle of application development covers the tasks required to start, complete, and maintain an application. The tasks start with planning and conclude with maintenance activities.

To help eliminate errors and misunderstandings, you need a structured language to depict the requirements. It must be something that everyone understands and can be interpreted in only one manner. You use data modeling to create this structure.

Data modeling shows how things (entities) are related and interact with each other. Data modeling techniques are very structured and defined so that an idea can be represented in only a single way. This approach enables you to eliminate ambiguities.

CA 2E uses data modeling to capture and describe application specifications. These specifications reflect the user's requirements. Once these requirements are represented by the data model and agreed to by the users, the other tasks in the application life cycle will use this information to develop the application. With correct information in the data model, many errors and changes will be eliminated in the later stages of the life cycle. This makes for a more efficiently developed application.

Advantages of a Data-Driven Approach

With a process-driven approach, you define your functions and programs first and try to fit the data required into the processes. With a data-driven approach, you begin with defining the data first. Each data element is defined once and only once.

A data-driven approach eliminates data redundancy and sets the stage for normalizing the database, making it easier to be accessed and maintained.

Defining a data model helps insulate the data structure from the process logic. That is, when process logic or flow changes, the data structure or model does not need to change. Since process logic is more susceptible to change than the data model, the structure is easier to maintain. Using an Order Entry example: if the Order information is defined by the order entry transaction, the database is likely to change when the transaction changes. On the other hand, if the order information is defined based on all the information that users think of regarding an order, the data structures are likely to accommodate any set of transactions.

Your data model restricts the number of allowable operations and suggests processes that can operate on the data. Some database functions are common to all entities regardless of their structure. For example, editing a Customer entity is the same process as editing a Supplier entity although they have different entity attributes. Inquiry and reporting processes are possible for all entities.

This set of processes leads to process structures that can become building blocks for larger, more complex processes. By reducing or eliminating redundancy, processes are easier to manage and maintain.

The CA 2E Approach to Data Modeling

There are several different methodologies for creating and analyzing data models. These methodologies differ considerably in the degree of rigor and formality of their approaches, and in the terminology used.

CA 2E takes a data-driven approach to data modeling: it uses the entity-relation (E-R) modeling method. An E-R model lists the significant business objects of an organization and the relationships between these objects.

The objects are called entities. Each entity has its own properties, called attributes, that distinguish it from another entity. For example, a company, person, or product is an entity; a company ID, a person's name, or a product code is an attribute.

The company entity can be described and uniquely identified by its attribute; for example, company ID. Other examples of unique identification might be a person by the person's name, or a product by a product code.

In CA 2E, an entity means a file, and an attribute means a field on the file. The associations between files or between a field and its file are called relationships.

CA 2E uses the basic English verbs, *Known by*, *Has*, *Refers to*, and *Owned by*, to represent these types of relationships. Relationships drive the design process. CA 2E is able to create the database for an application system entirely and automatically from the types of relationships it recognizes in a data model.

In addition, CA 2E uses what it knows about relationships and integrity rules stored in a data model to automatically generate default programs to implement an application.

For more information about entities, attributes, and relationships, see the chapter "Developing a Conceptual Model."

For more information about types of CA 2E relations and their use, see Chapter 3, "Understanding Your Data Model."

The following example illustrates a CA 2E data model for a simple Order Entry application. The model identifies the entities, and their relationships, and the types of the relationships using four basic relations.

To describe data models, CA 2E uses a simple data modeling language consisting of statements of the form:

<Subject Verb Object>

Example: A Customer has a Customer name

Example of a CA 2E Data Model

Following is an example of a data model used by CA 2E:

A Customer is **Known by** a Customer code

A Customer **Has** a Customer name

A Customer **Has** an Address

A Customer **Has** a Credit limit

A Product is **Known by** a Product number

A Product **Has** a Product price

An Order is **Known by** an Order number

An Order **Refers to** a Customer

An Order **Has** an Order status

An Order **Has** an Order date

An Order detail is **Owned by** an Order

An Order detail is **Known by** a Line number

An Order detail **Refers to** a Product

An Order detail **Has** an Order quantity

Chapter 2: Developing a Conceptual Model

This chapter provides direction for developing a conceptual data model based on the analysis of business information and requirements of your application project.

This section contains the following topics:

Before You Begin

You should be familiar with entity relationship diagramming conventions in order to present the relationships with an ERD.

Overview

You use the conceptual data model to identify and record the specific data elements needed to store and retrieve information. The data and data connections described by your model will be used to create files in the database serving your application system. Your conceptual model will be used to create a model later in CA 2E.

You can build a conceptual data model with pencil and paper. You can also use a variety of design tools to develop your data model and bring it into CA 2E. If you use design tools, see Considerations topic in this chapter.

In this chapter, you will

- Identify the entities, attributes, and relations to represent the business information you want to describe in your data model.
- Normalize your model to prepare for entering it into CA 2E.

When you have successfully completed these tasks, you are ready to use CA 2E to create a data model or to maintain an existing one.

See the chapters "Understanding Your Data Model" and "Creating/Defining Your Data Model" for more information.

See the chapter "Introducing Data Modeling" for more information about basic concepts on data modeling and examples of a CA 2E data model.

Goals of Your Conceptual Model

Your conceptual data model should be built so that it can be effectively used to produce a CA 2E data model. This model will then be generated as a database for your application.

Your conceptual data model should be:

- Comprehensive

Every item of information that is relevant to the organization should be recorded. Every item of information should appear once and only once.

- Consistent

There should be no mutually incompatible representations of information, inconsistencies, or conflicting rules about what can be done with the data.

Identifying Entities and Attributes

This is the first task you perform to begin building a conceptual data model. The purpose of this task is to produce a list of primary entities and the attributes for those entities.

This task consists of two steps:

- Step 1 shows you how to identify the primary entities for your model. This step also discusses generalization and differentiation of the entities.
- Step 2 shows you how to identify the attributes for your primary entities.

Although you can identify the entities, attributes, and their relationships at the same time, it may be easier to follow the steps and examples in the order they are listed.

To guide you through and give you a complete picture of the process, a sample of an Order Entry application and examples will be used for this task.

The same examples will be used again in the next task, Identifying Relations.

Step 1: Identifying Primary Entities

The first task in building your conceptual data model is to review the data items from your analysis of business information. Determine which are the most pertinent items and group them into primary entities.

To determine primary entities, select the objects that are important to your business. The end users of your application can be a good source to help you identify primary entities. They know what the entities are for their particular business because they physically handle them everyday.

Customers place orders for products on order entry forms. The company sells products. Your application needs data to process an order entry form to meet the business requirements. The following example shows a typical Order Entry form.

```
ABC Company                                          Customer Copy

Customer Code:

                                                    Phone:

Name

Address                                              Order Number

                                                   Date


Product Code          Description     Qty            Price          Total

__________            __________      ___            _____          ______

__________            __________      ___            _____          ______

__________            __________      ___            _____          ______

__________            __________      ___            _____          ______
```

For this Order Entry application, you would consider the following primary entities. They are represented as independent boxes:

Order	Product

The entities you have just selected (Customer, Company, Order, and Product) represent information that is pertinent to this kind of business.

However, you realize there will be situations when some products being ordered may be out of stock. Those particular products must be put on purchase orders, which will be placed with specific vendors or suppliers who supply them to the company. Each vendor or supplier supplies a particular type of product, and you want to be able to distinguish between products.

You now want to add more entities to represent your requirements:

Customer	Company
Order	Product
Vendor	Purchase Order
Supplier	Supply Request

Note: Entities should be given precise and concrete names. For example, use Customer instead of Person, or Country instead of Place. Use singular instead of plural form, such as Customer, not Customers.

You have identified and listed some entities you think are important to represent and describe the information needed for your business application. At this point, the list of entities you produced is the first version of your data model.

Your entities should be constantly re-evaluated, reorganized, and redefined to ensure that they are really what you meant them to be, and that they can be used as you intended.

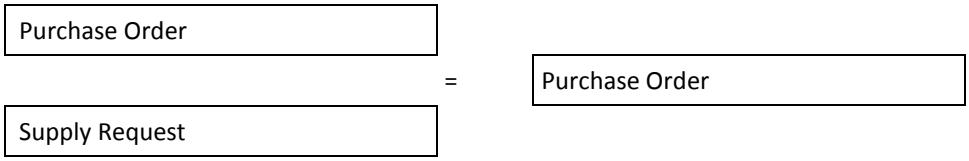
Generalization and Differentiation of Entities

You can refine the information, represented as entities, in your data model as often as needed to meet your data requirements while designing the database for your application. This topic explains how you use the two opposite processes called "generalization" and "differentiation" to refine your identified primary entities.

Generalization and differentiation involve deleting or adding entities to your model. At this early design phase, these processes affect the number of files that will have to be created for your application. Generalization and differentiation mean naming and renaming the entities and attributes, and assigning and reassigning different attributes to different entities.

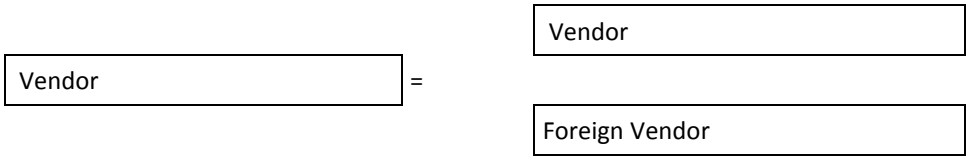
With generalization, you combine two entities representing different types of the same thing into one entity. With differentiation, you divide one entity into two separate ones because you decide the entity is actually representing two different things, and should be divided.

Going back to the Order Entry example discussed earlier, you may want to review the Purchase Order and Supply Request entities. Do you really need both Purchase Order and Supply Request? Are they both representing the same thing: a customer's request for a product not in stock, which the company will have to purchase from a specific vendor or supplier? If you decide they are, then they can be generalized and combined into one entity:



The Vendor and Supplier can also be generalized and combined into one entity, Vendor, in the same way. To discuss differentiation, use the Order Entry example again and look at the Vendor entity.

Does Vendor represent all of the information about a vendor, or are there special vendors with special information? In this case, a vendor from a foreign country requires special import and duty information that a native vendor does not require. Because of this special information, you need to add a new entity:



Step 2: Identifying Entity Attributes

Attributes supply informational detail to the entity. An entity can have one or several attributes. Each instance of an entity must contain values for all the attributes to define it. For possible values an attribute can take, see the topic, Domains, at the end of this step.

Each attribute represents one characteristic of the entity.

Beginning with the Customer entity, you would consider the following its potential "attributes":

CUSTOMER
Customer Code
Customer Name
Customer Address
Customer Credit Limit
Customer Account Balance

For the Product entity, the attributes would be:

PRODUCT
Product code
Product Name
Product Type
Product Description

After identifying all the attributes for the primary entities of your Order Entry application, you have a list that looks like this:

Entity	Attributes
Customer	Customer Code
	Customer Name
	Customer Address
	Customer Credit Limit
	Customer Account Balance

Product	Product Code Product Name Product Type Product Description
Order	Order Code Order Quantity Order Line Vendor ID Customer Code Product Code
Vendor	Vendor ID Vendor Name Vendor Address
Foreign Vendor	Foreign Vendor ID Foreign Vendor Name Foreign Vendor Address Foreign Vendor Country Foreign Vendor Duty Percentage
Purchase Order	Purchase Order Code Purchase Order Quantity Purchase Order Line

Domains

The set of possible values an attribute can take is the *domain* of the attribute. When modifying your data and describing attributes, it is important to think of their domains. Each instance of an entity must contain values for all attributes that define the entity.

If an entity contains the attribute City, each instance of the attribute must draw its value from the domain of City. It cannot draw its value from any other domain.

Two attributes may have common characteristics (the same length, data type, and valid values) but not share the same domain. The attributes Customer Number and Order Quantity both may be 6-digit numeric fields. However, they draw their values from different domains, a Customer Number domain and an Order Quantity domain.

Other attributes may have common characteristics and share the same domain. For example, Order Quantity and Shipped Quantity are both 6-digit numeric fields. They share the same domain and draw from the same set of possible values.

In relational modeling, you can compare attributes that share the same domain. You cannot compare attributes that do not share a domain.

Identifying Relations

When you finish identifying the entities and attributes of your conceptual data model, you are ready to identify the relations. The purpose of this task is to produce an Entity-Relationship diagram (ERD) of your data model, showing the types of relationships that link the entities. The diagram eventually will be translated into a data model in CA 2E.

The following examples show how you can identify and represent relations in a data model. You can use any diagramming methodology.

This task consists of these steps:

- Step 1 explains how to identify relationships between entities or between an entity and an attribute.
- Step 2 describes how to select a primary key for an entity.

Data Relationship Connections

In your data model, a relationship establishes a connection between one entity and another or between an entity and an attribute. In the database, a relationship links a file to a file, or a file to a field (of the file).

You can identify and categorize relationship types by studying the connection to see whether it falls into the following categories, asking yourself questions such as:

Optional—Is the connection mandatory or optional?

- Does an order require a customer?
- Does a product require a description?
- Does a product require a price?

Cardinality—Is the connection a one-to-one, one-to-many, or many-to-many?

- How many customers can be on a single order?
- How many orders can a customer place?
- How many prices are there for a product?
- How many addresses can a customer have?

Involution—Does the connection exist between two different entities or from the same entity to itself?

- Are components a product, or are they themselves products in their own right?
- Is a manager an employee or is there a separate entity of just managers?

  If Manager is an employee, then the relation "Employee Works for Manager" connects to the same entity. This is called involution.

By answering those questions and using an ERD, you will be able to determine the relation types of your data.

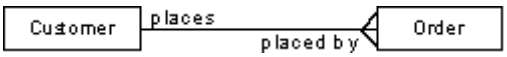
All relationships in CA 2E data model descriptions are mandatory. You can make relationships optional at the process level.

Step 1: Identifying Relations Between Entities

This step details relation cardinality, which is the number of entity instances for a relationship. This topic contains examples that illustrate different types of data relationships.

Use these examples as guidelines to identify your entity-to-entity connections and draw an ERD for your data model. Let us examine two separate entities in your model, Customer and Order, and find out what the significant connection between them is.

You can tell that a relationship exists between these two entities because the customer places the order and the order is placed by the customer, as illustrated in the following example of a common ERD.



This relationship is viewed from both perspectives: from a customer and from the order. Viewing a relationship from more than one perspective is important because this gives you more information about the nature of the relationship.

With CA 2E relations, the implementation of the data model is one-way, except for the Extended by relation.

Examples of Relationships

The following examples show one-to-one, one-to-many, and many-to-many relationships.

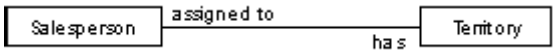
- One-to-One

  In a one-to-one relationship, each instance of one entity is related to one, and only one, instance of another entity.

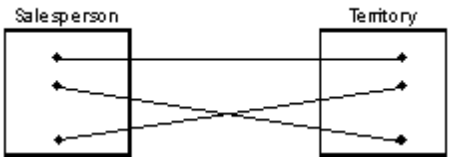
  For example, to facilitate quick service, each salesperson in the company is assigned to one territory. Customers in a territory are serviced by the salesperson assigned to that territory.

  The relationship between the two entities Salesperson and Territory is a one-to-one relationship because each salesperson is assigned to one territory and each territory has one, and only one, salesperson assigned to it.

  The following ERD shows this one-to-one relationship.



The following instance diagram shows a one-to-one relationship.

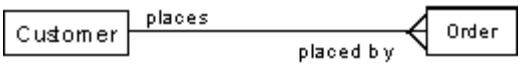


- One-To-Many

  In a one-to-many relationship, one instance of an entity is connected with several different instances of another entity.

  For example, a customer places more than one order with the company; an order is placed by a customer. The relationship between the Customer entity and Order entity is a one-to-many relationship because a customer can place more than one order with the company; however, an order can be associated with one, and only one, customer.

  The following ERD shows a one-to-many relationship.



  The following instance diagram shows a one-to-many relationship.

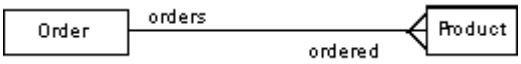
- Many-To-Many

  In a many-to-many relationship, an instance of one entity is related to more than one instance of the other entity at a time and vice versa.

  For example, the relationship between the Order entity and Product entity is a many-to-many relationship because:

  - An order can be placed for more than one product at a time.
  - More than one product can appear on a single order.
  - A product can appear on more than one order at a time.

  The following ERD shows a many-to-many relationship.



  The following

Step 2: Selecting Primary Key (Unique Identifier) for an Entity

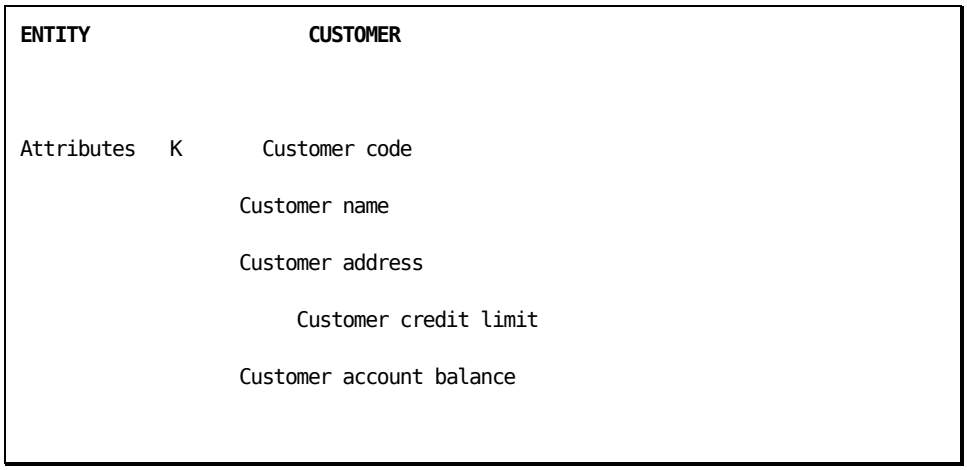
This step provides information for selecting primary keys for an entity.

- **Primary Key** or **Unique Identifier**—An attribute or group of attributes assigned to an entity to uniquely define an instance of the entity.
- **Foreign Key**—An attribute or group of attributes of an entity that connects this instance with an instance of another entity. In this way it defines the relationship between two entities. It consists of attributes defining the primary key of the related entity.

A primary key can either be a single attribute, a relationship, or a combination of attributes and relationships. Each entity must have a single primary key. The entity can have several alternate keys.

For example, the Customer entity in your Order Entry model may have several instances representing different customers (Customer A, Customer B, Customer C). An instance is equivalent to a record among other customer records in your Customer database file. The primary key uniquely identifies each record.

You select Customer Code as the primary key for Customer entity:



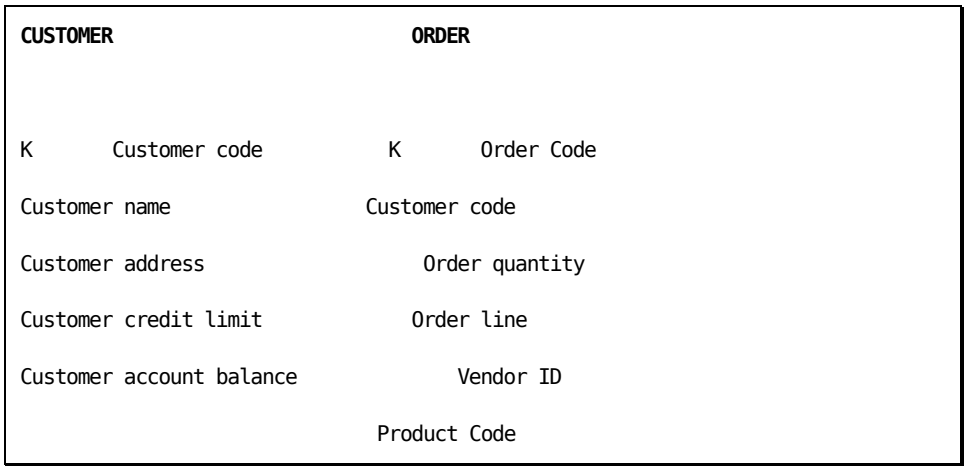
Although you could have chosen Customer Name as a key instead, a code is a better choice to ensure uniqueness. It also allows the customer's name to change without having to create a new customer record.

Note: Avoid using keys whose values can change.

A primary key can also be used to implement the relationship between entities by forming a link between the entities.

For example, in the case of Customer and Order entities, the link is Customer Places Order. The relationship between the two entities Customer and Order is recognized by the presence of the Customer Code in the list of attributes of the Order entity.

Example:



Customer Code is the primary key of the Customer entity; it does not play a role in identifying orders. However, Customer Code is needed in the Order entity to identify which customer placed the order.

In this case, the Customer Code is used to represent the relationship between Order and Customer. It becomes a foreign key in the Order entity file. The relationship means a single customer belongs to this order.

Implementing Entity To Entity Relationships

You implement entity relationships by using foreign keys.

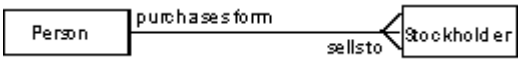
- The primary key of one entity, when used in another entity, provides the link between the two entities.

To implement a one-to-one relationship:

- Make the primary key of one entity the primary key of the other entity.

One-to-One Relationship

Example:




```
     Person

K Person ID

       Person name

     Person address




      Stockholder

K    Person ID
```

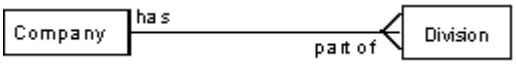
Number of Shares

To implement a one-to-many relationship:

- Use the primary key of entity A as a foreign key in entity B.

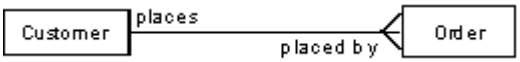
One-to-Many Relationship

Example 1:



```
 Company


K  Company code

    Company name

    Company president


 Division

K       Company code

K   Division code

   Division name

   Division president
```

Example 2:



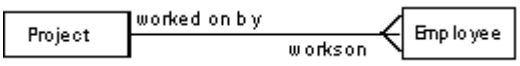
```
      Company


K  Customer code

   Customer name

   Customer type

   Customer credit limit

   Customer account balance


  Order


K    Order code

      Customer code

   Order quantity

  Order line

      Vendor ID

       Product code
```

To implement a many-to-many relationship between A and B:

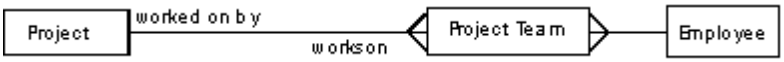
- Create new entity C to contain the primary keys of A and B.
- Remove the relationship between A and B.
- Add a one-to-many relationship from A to C and B to C, where the primary keys of A and B compose the primary key for C.

Many-to-Many Relationship

Example:



Change to:




```
Project                          Employee


K Project code                   K       Employee code

   Project name                           Employee name

   Product release                        Employee hire date


                        Project Team


              K       Project code

            K       Employee code
```

See the chapter "Understanding Your Data Model" for more information about relations.

Normalizing Your Data Model

Normalization is the process of removing data redundancy and duplication from the entities and attributes of a model.

This process involves:

- Regrouping attributes
- Splitting entities
- Reassigning primary keys

Before starting the process you need to understand data relationships and familiarize yourself with definitions of terms on key dependencies.

In order to understand the relationships among data items, you must determine which attributes of an entity are dependent on the entity's other attributes. Each entity must have a unique key by which it can be uniquely identified. The key can be a single attribute or group of attributes. A key must have two properties:

- In each instance of an entity, the value of the key must uniquely identify that instance.
- If the key is composed of more than one attribute, each of the attributes must be essential to the unique identification of the entity.

Functional Dependence

Functional dependence describes the relation between the key and non-key attributes of an entity. An attribute of an entity is functionally dependent on a key of that same entity if, for each value of the key, there is one and only one value of the non-key attribute.

The non-key attributes are functionally dependent on Supplier Number because there is only one precise corresponding value for Name, City and Postal Code for a particular value of Supplier Number.

Full Functional Dependence

Full functional dependence further qualifies the relationship when the entity key is composite; that is, composed of multiple attributes. Full functional dependence occurs when non-key attributes are dependent on all the key attributes, not just some of them.

The following is an example of functional dependence:

```
Supplier


K        Supplier number

 Supplier name

   Supplier city

   Supplier postal code
```

Full functional dependence further qualifies this relationship. For example, this term states that the attribute(s) N of entity T is fully functionally dependent on the attribute(s) K of entity T, if N is functionally dependent on every attribute of K but not on any subset of K. This means that if an entity is uniquely identified by more than one attribute (a composite key), each of its non-key attributes must be functionally dependent on the entire key. If an attribute is dependent on a subset of the key, then such dependency indicates that the attribute belongs in an entity having that subset of the key as its unique identifier.

In the following example, the non-key attributes are not fully functionally dependent on a composite key:

```
Supplier


K        Supplier number

K        Supplier status

 Supplier name

   Supplier city

   Supplier postal code
```

When the Supplier Status is introduced as part of the primary key, the other non-key attributes are not fully functionally dependent on the key. The non-key attributes can be functionally dependent on just the Supplier Number. For example, the Supplier Name is the same regardless of the Supplier Status value. The non-key attributes are functionally dependent on the primary key but they are not *fully* functionally dependent on the whole key.

If the non-key attributes are functionally dependent but not fully functionally dependent, the primary key must be a composite.

A composite key can have fully functionally dependent attributes. For example, you might want a composite key of Date and Time, where both are necessary and required. All of the attributes will be fully functionally dependent on the composite key:

```
Transaction



K       Transaction date

K       Transaction time

        Transaction quantity

    Transaction type

        Transaction product
```

Normalization is the last task you perform to finalize your conceptual data model before entering it into . Now that you have completed an ERD for your data model, you can begin this task.

The purpose of this task is to help you refine your entities and attributes to arrive at a third normal form by applying the three normalization rules: first normal form (1NF), second normal form (2NF), and third normal form (3NF).

During normalization, an unnormalized entity can be analyzed, reorganized, and progressively transformed into new entities. The process is reversible, therefore no information will be lost during transformation.

This task consists of the following steps:

- Step 1 covers creating first normal form (1NF).
- Step 2 covers creating second normal form (2NF).
- Step 3 covers creating third normal form (3NF).

Step 1: Creating First Normal Form (1NF)

First normal form (1NF) is the process of eliminating repeating data groups. This is done by representing data in the form of more than one entity. The remaining part of the normalization process analyzes entities and attributes in terms of functional dependence.

The rule states that for a primary key, there will be only one value for each non-key attribute.

Use the Order entity as an example. This is the Order entity before normalization:

```
Order


K    Order number

K   Customer code

   Customer name

   Customer phone number

   Customer address

        Customer postal code

    Order date

      Product code            }

       Product description     } These attributes

      Product size            } can have up to

        Product quantity               } 4 sets of values.

     Product price           }

Order detail total     }

       Order total
```

Review the Order entity against the Order Entry Form, described under Step 1: Identifying Primary Entities.

This entity contains a repeating group of attributes: Product Code, Product Description, Product Size, Product Quantity, Product Price, and Order Detail Total.

Remove this repeating group of attributes.

Your Order entity now contains only these attributes:

```
Order


K  Order number

K   Customer code

   Customer name

   Customer phone number

   Customer address

        Customer postal code

    Order date
```

Create a new entity and place the group of attributes you removed from the Order entity into this new entity:

```
Order Detail


K        Order number

K   Product code

    Product description

     Product size

    Product quantity

        Product price

   Order detail total
```

Step 2: Creating Second Normal Form (2NF)

An entity is in second normal form (2NF) if it is in first normal form and every non-key attribute of this entity is fully functionally dependent on its primary key.

The following entity is in 1NF but the non-key attributes are not fully functionally dependent on the primary key.

```
Supplier's Inventory


K   Supplier number

K        Part code

       Supplier name

   Supplier status

 Supplier status description

     Part Quantity
```

SNumber	PNumber	Name	Status	Sstatus Desc	Part Qty
111	05	Computer Store	105	Wholesale	100
111	10	Computer Store	105	Wholesale	210
111	15	Computer Store	105	Wholesale	534
245	05	Floppy Discount	100	Wholesale	498
245	10	Floppy Discount	100	Wholesale	021

You can see that the Supplier Name and Status are functionally dependent on Supplier Number only and not the Part Number. Therefore, the Supplier Name and Status are not fully functionally dependent on the Supplier Number and Part Number. The entity is not in 2NF.

The Supplier Status is functionally dependent on Supplier Number. This dependency means that a value of Supplier Number requires a specific value of Supplier Status. If the Supplier Number is 111, the Supplier Status must be 105.

Make the following changes to get the entity into 2NF:

```
Supplier


K      Supplier number

 Supplier name

   Supplier status

 Supplier status description


        Supplier’s Inventory


K   Supplier number

K        Part number

     Part quantity
```

Both entities are now in at least 2NF. Supplier's Inventory is in 3NF.

Step 3: Creating Third Normal Form (3NF)

An entity is in third normal form (3NF) if it is in second normal form (2NF) and each of its non-key attributes is not dependent on another non-key attribute.

The Supplier entity is in 2NF and must be put into 3NF:

```
Supplier



K     Supplier number

 Supplier name

   Supplier status

 Supplier status description
```

Supplier Status Description is functionally dependent on Supplier Status. Third normal form requires that the entity Supplier be split into two entities: Supplier and Supplier Status. You define the entity Supplier as follows:

You then define Supplier Status, as follows:

```
Supplier Status



K    Supplier status

 Supplier status description
```

You can now maintain Supplier Status information separately. A change to a Supplier Status Description does not require an update to the Supplier instance.

With third normal form, all attributes in an entity are fully functionally dependent on only the entire key of the entity.

CA 2E Basic Relations

CA 2E provides four basic relation types to describe data relationships: Owned by, Refers to, Known by, and Has.

A relationship expresses an association between two files, or between a file and a field. It constitutes the fundamental links in a CA 2E data model. Such links enable you to explicitly assert the meaning of the connections within your data. A file's list of relationships will be automatically resolved into the fields needed to implement that file.

File-to-field Relationships

To describe a file-to-field relationship, you may use:

- Known by to declare a field to be present on a file as a key field.
- Known by to declare a field to be present on a file as a data field.

File-to-file Relationships

To describe a file-to-file relationship, you may use:

- Refers to specify an association between two mutually independent files. It causes the key entries of the referenced file to be included as foreign key entries on the referring file.

  For example, if Order Refers to Customer, then the keys of Customer are included as foreign key entries in the Order file:

```
  Order          Refers to             Customer


K Order code                     K      Customer code

   Order quantity                       Customer name

   Order line                           Customer address

        Vendor ID                            Customer credit limit

   Customer code                        Customer account balance

        Product code
```

  In CA 2E, a Refers to relationship has a one-to-many cardinality.

- Owned by to cause the key entries of the owning file to be included among the key entries of the owned file. It specifies that the high order keys of the owned by file are the keys of the owning file.

  For example, if Order is Owned by Company, then the key(s) of the Company file appears as the major key(s) on the Order file:

```
  Order          Owned by                    Company


K  Company ID                    K      Company ID

K     Order code                          Company name

    Order quantity                       Company address

 Order line

       Vendor ID

       Product code
```

In , the Owned by relationship has a one-to-many cardinality: the (Owned by) Order entity is the child of the parent (Company) entity.

Considerations

This topic covers some performance, assimilation, and design tool considerations that may help you when developing a CA 2E data model.

Performance

Since your data model's entities and attributes become files and fields in the database, the way you design your model affects the number of times your application system needs to access information to carry out a business transaction.

A fully normalized database breaks the data down into more files than a partially normalized one does. It, therefore, requires more input/output activities to access several files during processing.

For performance reasons, you may consider violating 3NF. You may define files that are frequently accessed as 2NF, such as files with highly volatile records, inventory records, and work records. The 2NF files contain a level of redundancy.

Existing Database

You can integrate your existing files or database into your CA 2E data model through a process called assimilation. Existing files and field names are maintained to ensure that both new and existing systems can use the same files.

See the chapter "Assimilation" for more information on using existing i OS files.

Using Design Tools

Instead of creating a conceptual data model with pencil and paper you can also use a variety of design tools.

For example, CA Xtras Gateway (GWY) provides a bi-directional bridge between SILVERRUN RDM and CA 2E data models that lets you import and export design specifications. This includes full support for the following CA 2E model objects: files, fields, relation, and narrative text.

Chapter 3: Understanding Your Data Model

This chapter introduces CA 2E files, fields, conditions, and relations, and how these relate to your conceptual model.CA 2E files, fields, conditions, and relations are the basic building blocks you will use to define and maintain your data model.

The data model you define serves as a foundation for developing a correct database for your entire application system. It describes the files, fields, conditions, and their relations. The data model also includes the validation rules and edit codes to be used by your application system.

This section contains the following topics:

CA 2E Data Model

A CA 2E data model is made up of a number of design elements, or CA 2E model objects. These are building blocks that can be put together according to certain rules to design your data model. For example:

- **File—**contains a list of relations that will be resolved into a list of file entries. A file represents an entity in the model.
- **Field—**describes an item of data.
- **Condition**—describes the values or set of values (domains) that a field may take.
- **Relation**—describes a connection between two files or between a file and a field.

CA 2E Data Model Objects

CA 2E objects are not i OS objects. A CA 2E object exists only within a design model.CA 2E data model objects are classified into different types: files, fields, conditions, and relations.

CA 2E objects are interrelated in a data model design as described below:

- A file can reference either another file or a field through a relation.
- Conditions are attached to fields to specify the values that a field may take.

CA 2E objects are referred to by an object name.CA 2E object names must be unique. Specifically:

- A file name must be unique within the entire model.
- A condition name must be unique within the based-on field.
- A field name must be unique within the design model.

CA 2E makes no distinction between upper and lowercase characters in an object name. You can specify an object name in different ways and CA 2E will treat them as being the same. For example:

- Order code
- ORDER CODE
- Order Code
- ORDer coDE

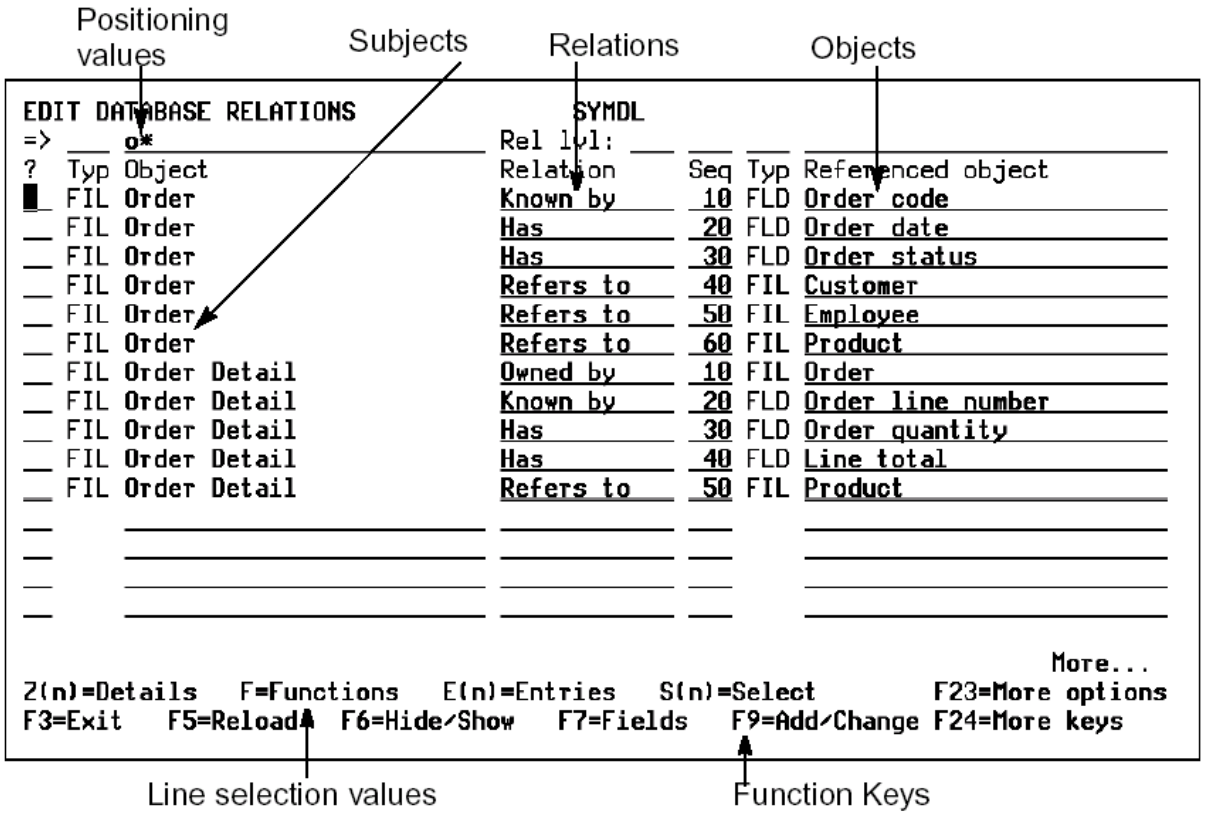
From Your Conceptual Model to a CA 2E Data Model

Your conceptual model contains the terms that have the following equivalents in a CA 2E data model:

Conceptual Model	Data Model
Entity	File
Attribute	Field
Relationship	Relation
Domain	Field conditions; Reference field types

Edit Database Relations Panel

You will use the Edit Database Relations panel, as shown below, to describe your data model to CA 2E. See the chapter “Creating/Defining Your Data Model” for more information about this panel.



The rest of this chapter describes CA 2E files, fields, conditions, and relations in detail.

Using Files

This topic provides conceptual information about files and a full description of the different file types, with examples of how they are used.

CA 2E Files

A CA 2E file represents an entity within a CA 2E model; for example, an Order or a Customer. A CA 2E file is defined by CA 2E relations. CA 2E automatically resolves the relations to determine which fields are to be present on a file. The presence of a field on a file is called an entry.

Properties of CA 2E Files

Each CA 2E file has a name and a file type. It also contains other descriptive details such as a documentation sequence and whether the file is retrieved from an existing i OS file.

In addition, each CA 2E file has two default messages associated with it. Default messages are created for each type of CA 2E file. These messages appear when you attempt either to access a record that cannot be found in the file or to add a record to a file for a key value that already exists.

File Name

CA 2E file names must be valid CA 2E names and unique within the model. A file name can contain up to 25 alphanumeric upper or lowercase characters including embedded blanks.

File Type

A file type is determined by the intended use of the file. File types are listed in the table below.

File	Attribute	Description	Example
Database	CPT	Capture file	Order entry file
	REF	Reference file	Company file
Non-database	STR	Structure file	Audit date and time stamp

Capture (CPT) and reference (REF) files are database files; data structure files are non-database files. Capture and reference files are resolved into i OS files for implementation.

A structure file cannot stand by itself as an i OS file. It defines a structure of fields that can be included in more than one file.

Reference (REF) Files

Reference (REF) files are master files that typically contain non-volatile information.

Examples of REF files include:

- Customer
- Product
- Area code
- Location code
- Currency

Capture (CPT) Files

Capture (CPT) files typically contain transactional data that is recorded regularly for use by your application.

The CPT file type is given to files that record high volumes of transactions and require constant update. CPT files generally refer to reference entities for supporting information.

Examples of CPT files include:

- Order
- Transaction
- Ledger Entry

Default Functions for REF and CPT Files

REF and CPT files have different default functions. CA 2E automatically defines a number of standard functions for files that you create for each of these two specific file types.

Default Functions for REF Files

Function	Associated Access Path
CRTOBJ - Create Object	UPD
DLTOBJ - Delete Object	UPD
CHGOBJ - Change Object	UPD
SELRCD - Select Record	RTV
EDTFIL - Edit File	RTV

Default Functions for CPT Files

Function	Associated Access Path
CRTOBJ - Create Object	UPD
DLTOBJ - Delete Object	UPD
CHGOBJ - Change Object	UPD

STR Files

An STR file contains a group of fields. These fields can be incorporated into a number of other files through the use of the Includes font relation. STR files define data structures that are used in several places in your data model. Because STR files are not database files, access paths cannot be specified for them.

Example of an STR File: Audit Stamp

CA 2E File versus i OS File

A CA 2E file is similar to an i OS file in the sense that it is a list of fields. A CA 2E file is different from an i OS file because it has relations specified for it. These relations specify referential integrity checks to be performed in the functions that use the file. The i OS Database Manager does not perform referential integrity checks between database files.

CA 2E creates the necessary source code to perform referential integrity checking. This is validation of the relations between files. As an example, use the entity Employee is Owned by the entity Company. When a Company Code is associated with an Employee Code, CA 2E checks to ensure that the Company Code exists in the Company file.

CA 2E lets you specify whether or not these integrity checks should be performed. This is done by specifying whether the relation is mandatory or optional. If the relation is mandatory, the end user will have to enter a valid value in the foreign key field. If the relation is optional, the end user has a choice whether to enter a value; however, the value entered must be valid.CA 2E also lets you specify your own checking process if you desire.

See the chapter "Modifying Device Designs" in *Building Applications* for more information on mandatory and optional checking.

Using Fields

This topic provides conceptual information, a full description of different field types, and examples of how they are used within your model.

CA 2E Fields

A CA 2E field represents an attribute within a CA 2E model; for example, Customer Code, Order Number, or Product Price.

Fields that are placed in a file from the resolution of CA 2E relations are called entries.

See the chapter "Creating/Defining Your Data Model" for more information.

Properties of CA 2E Fields

A CA 2E field has a field name and field type.

Field Name

A field name must be unique within the data model. It can contain up to 25 alphabetic characters in upper or lowercase, and numeric characters, including embedded blanks.

The field name is the title of the field and not the implementation name that CA 2E assigns for each field.

Field Type

A field type indicates which specific types of values can be entered for a field.

CA 2E uses field types to:

- Determine default attribute values for the field such as length, validation, edit code
- Prevent operations from working with mixed field types
- Impose integrity checking rules for data input validation

Default field types are defined in the CA 2E shipped file, *Field Attribute Types. You can add your own field types. Following are the shipped default field types.

- Alphanumeric Fields
  - CDE (code)
  - DT# (ISO date)
  - IGC (ideographic text)
  - NAR (narrative text)
  - TM# (ISO time)
  - TS# (ISO timestamp)
  - TXT (descriptive text)
  - VNM (system name)
- Numeric Fields
  - NBR (number)
  - VAL (value)
  - QTY (quantity)
  - PRC (price)
  - PCT (percentage)
  - DTE (date)
  - TME (time)
  - SGT (surrogate)
- Special Fields
  - STS (status)
  - REF (reference)

See the chapter "Maintaining Your Data Model" for more information on user-defined field types.

Field Attribute Values

CA 2E provides a number of default attribute values for fields based on the field type given to that field:

- **Basic attributes**—Characteristics such as length and implementation name. For example, by default, quantities (QTY) are numeric and seven digits long.
- **Text**—Several different types of text can be associated with a field. The text is used to document fields and title fields on device designs.
- **Validation attributes**—These attributes specify how data entered into the field is to be validated. Validation can include attribute checking such as upper/lowercase checking, Modulus checking, valid name checking, and validation through a check condition.

Overriding CA 2E Default Field Attributes

Some of the default values of the field types may be overridden, both at the field level and the device design level:

- **Field values**—Describe the attributes of each individual field in the data model. They are used as the initial defaults when new fields are created. You can change these values with the Edit Fields panel.
- **Device field values**—Describe the attributes of each individual instance of the field on a device design. You can override some attribute values, such as the field heading text and validation conditions, using the Edit Device Field panel.

Field Usages

CA 2E fields may be categorized as database fields or function fields.

Database Fields

Database fields include:

- Key fields to identify files
- Attributes to represent non-key fields on a file

Database Field Usages

Type	Description	Example
CDE	Code (key)	Company code
ATR	Data attribute	Company name

Key database fields are given the usage type of CDE (code); attribute database fields have the usage type of ATR (attribute). CA 2E automatically supplies the usage type according to how the field is first used in a relation. You can change the usage type using the Edit Field Details panel.

If the field has a usage of CDE and is being associated with a file that has an attribute relation (Has), a warning message is issued that the usage is different. This is also true if a field usage is defined as ATR, and the field is the referenced object of a key relation (Known by, Qualified by).

Function Fields

A function field is only used in functions and does not reside in a database file.

You can add function fields directly to the field dictionary and then use them in device designs and action diagrams.

Function fields have six different types of usages. A function field usage may be one of the standard types or a user-defined function. Following are the function field usage types:

- CNT (count)
- DRV (derived)
- MAX (maximum)
- MIN (minimum)
- SUM (sum)
- USR (user-defined)

The SUM, MIN, MAX, and CNT field usage types provide standard field level functions. For example, SUM and CNT, for summing and counting; MIN and MAX, for specifying a minimum or maximum numeric value.

The USR and DRV field usage types allow you to define your own function field.

For more information, see Using Function Fields, later in this chapter.

Defining a Field as a Data Area

Before entering your model, create a data area in the appropriate library by using the IBM Create Data Area (CRTDTAARA) command. You can then add a field defined as a data area to the *Standard header/footer file. On the Edit Field Details panel, specify the Type as TXT and the Default Condition as *DTAARA. Modify the four-character data area name in the DATAARA field to match the name of your data area.

For the *Standard header/footer file, CA 2E automatically initializes the field with the information from your data area. The data area must be in your library list when you execute your application.

To use an existing data area within your CA 2E applications, define an EXCUSRSRC function with an access path of *NONE. Define a BOTH parameter using a USR work field. This USR field must have the same attributes and length as the data area. In the EXCUSRSRC, add your own logic to access the data area and place the contents of the data area into the parameter field.

Shipped CA 2E Field Types

The definitions for all CA 2E field types (shipped and user-defined) are included in a shipped file called *Field Attribute Types. This file comes with the shipped version of CA 2E, but each model has its own copy. You can access this file from the Edit Database Relations panel. Changes made here apply to the model, not to the product or other models.

Displaying Existing Field Types

To display the field types:

1. Zoom into the *Field Attribute Types file by placing a Z against one of the file's relations on the Edit Database Relations panel. The Display Field Types panel lists the field types.
2. Zoom into a specific field type by placing Z against the field type to view the default values
3. Press F9 from the Display Field Types panel to add your own user-defined field types.

See the chapter "Maintaining Your Data Model" for more information on defining and adding your own field types.

The following table details the field types shipped with CA 2E.

Field Type Name	Description	Type	Internal Length	Example
CDE	Code	A	6	Stock code
DT#	ISO Date	A	10	Order date
D8	8-digit Date	P	8.0	Order date
DTE	Date	P	7.0	Date of birth
IGC	Ideographic text	A	20	Kanji name
NAR	Narrative text	A	30	Comments
NBR	Number	P	5.0	Number of employees
PCT	Percentage	P	5.2	Profit margin
PRC	Price or tariff	P	7.2	Unit price
QTY	Quantity	P	5.0	Stock quantity
REF	Reference	–	–	Field based on another
SGT	Surrogate	P	7.0	System key
STS	Status	A	1	Discontinued/Current
TM#	ISO Time	A	10	Time process starts
TME	Time	P	6.0	Transaction time
TS#	ISO Timestamp	A	26	Transaction date/time
TXT	Descriptive name	A	25	Product name
VAL	Monetary value	P	11.2	Stock value
VNM	Valid system name	A	10	File name

The following sections describe how to use the field types listed in this table.

Field Type Default Characteristics

Each field type has its own default characteristics. For each individual type listed in this topic, the default characteristics are specified. Following is an example of default characteristics for a field of type DTE.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxDT	–	Y	N
System data type	Packed	N	N	N
External length	6.0	N	N	N
Internal length	7.0	N	N	N
Decimal places	N	N	N	N
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Date	Y	Y	Y
Keyboard shift	N	N	N	N
Allow lowercase	–	–	–	–
Mandatory fill	–	Y	Y	N
Valid system name	–	–	–	–
Mod 10/11 check	–	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	–	–	–	–
Field exit option	Blank	Y	N	Y
Edit codes: Input	4	Y	Y	Y
Output	Y	Y	Y	Y
Report	Y	Y	Y	–

Implementation name

This field type is the suffix used to identify a field when generating implementation names. CA 2E will generate default implementation names if you have specified that you wish it to do so.

See the chapter "Using your Development Environment" in the *Administration Guide* for more information.

The field implementation name must be unique within your data model. This means that it can be used for only one field.

When the field name for each field is generated, the suffix specified here is appended to the generated prefix for that field.

For a field of type DTE, the suffix is DT. If you specify a suffix of MY for a user-defined field type, CA 2E generates an implementation name of AAMY.

System data type

This field type is the data type allowed for a field in the i OS file. It can be alphanumeric or numeric.

System data type is the data type to be stored in a database file or displayed on device files. It can have one of the following values:

A—Alphanumeric

P—Packed numeric

S—Signed numeric (Zoned)

B—Binary (not generated)

F—Floating

A and S are data types to be displayed on device files. These values default according to the field type. For example, DTE is numeric, TXT is alphanumeric. Values may only be changed for fields with numeric field types (DTE, NBR) or for user-defined field types for which the programmer/designer is allowed to change the data type.

External length

This field type is the length of the field displayed or printed on a report. This is the number of characters or digits allowed for the field on display panels or print files. Fields with decimal positions are entered as total number of digits, number of decimal places. For example, for a field to contain 999.99, the length would be 5.2.

See the chapter "Maintaining Your Data Model" for more information on defining a field type with differing lengths between external and internal fields.

Internal length

This field type is the length of the field when it is stored in a file. This is the number of bytes used to store a field in a file. Fields with decimal positions are entered as total number of digits, number of decimal places. For example, for a field to contain 999.99, the length would be 5.2.

See the chapter "Maintaining Your Data Model" for more information on defining a field type with differing lengths between external and internal fields.

Decimal places

This field type is the number of decimal places for numeric fields.

LHS text

This field type is the left hand side text used for the field heading. This is the text to be placed before the field on the same line as its heading on a display panel or print format.

Example:

Size code—BBBBBB

Quantity—BBBBBB

RHS text

This field type is the right hand side text used for the field heading. This is the text to be placed after the field on the same line as its heading on a display panel or print format.

Example:

Size code—BBBBBB (SMALL/MEDIUM/LARGE)

Column headings

This field type is the column heading text to be placed above the field on a display panel or print format.

Example:

Size code	Quantity
BBBB	9999.99
BBBB	9999.99

Keyboard shift

This field type specifies which keyboard shift is allowed for the field on panel files. It can have one of these values:

Blank—no keyboard shift

X, A, N, W, I, D, M—for alphanumeric fields

N, S, Y, I, or D—for numeric fields

O, J, E, W, G, or A—for ideographic fields

For more information on keyboard shift values, refer to the IBM *DDS Reference*.

Allow lowercase

This field type specifies whether the field values may be in lowercase. It can have one of these values:

Y—lowercase allowed

Blank—lowercase not allowed

Lowercase applies only to alphanumeric fields.

Mandatory fill

This field type specifies whether the field requires mandatory fill. This can have one of these values:

Y—mandatory fill

Blank—no mandatory fill

Valid system name

This field type specifies whether the field requires the valid system name check to be performed. Value entered for the field must be a valid i OS system name.

A valid system name must start with a letter, no more than ten characters long, and contain only letters, digits, or one of these characters "-", "#", "$", or "@".

Modulus 10/11 check

This field type specifies whether the modulus 10 or 11 check is to be performed. The value entered for the field must meet a modulus 10 or 11 check as specified by the DDS CHECK keyword. This can have one of these values:

- **10**—apply modulus 10 check
- **11**—apply modulus 11 check
- **Blank**—do not apply modulus check

Modulus check applies only to numeric fields.

Default condition

This field type is the default value to be used for the field when adding records to the database if no value is supplied.

Note: The default condition does not specify a default value for fields on display files or reports.

Default condition has these values:

- * NONE—no default condition
- **condition name**—condition that supplies the default value
- *** DTAARA—**indicates that the value of the field is to be retrieved from the data area specified in the DTAARA field. This field will appear when you type *DTAARA in the Default Condition field and press Enter.

The name of the data area must be a valid system name. The data area name is used as an internal field name within CA 2E, in place of the generated code name. In a program, the generated code name is used. The field is loaded from the data area at the start of the program.

An example of using data area name is the CA 2E shipped field *Company name. The value for this field is retrieved from a data area called YYCOTXA:

Default Condition = *DTAARA

Dtaara = YYCOTXA

Implementation name = CMP

Check condition

This field type is the name of the list condition used to specify check values for the field. The value for Check condition is:

- ***NONE**—no check condition
- **Condition name**—used to check the input value of the field.

Translate values

This field type specifies whether value mapping is required to translate the display value entered for a field into a different storage value, and vice-versa. This applies only to STS fields or user-defined fields for which value mapping has been specified. Allowable values are:

- **Y**—use value mapping
- **Blank**—do not use value mapping

Note: STS fields with translate values are implemented only in functions that have interactive displays; they are not implemented in PRTFIL or CA 2E internal functions.

Field exit option

This field type specifies whether there is a field exit value. Allowable values are:

- **Y**—field exit is required
- **Z**—right adjust, zero fill
- **B**—right adjust, blank fill
- **R**—right to left support for non-numeric fields

Edit codes

This field type specifies the edit codes for the panel input field, the display field, and the report field. The edit codes are specific to CA 2E,and may not relate to DDS edit codes.

The following table lists edit codes and their allowable values.

Edit Code	Description
–	For date fields: *mm-dd-yyyy or yyyy-mm-dd* For timestamp fields: *mm-dd-yyyy-hh*:*mm*:*ss* or *yyyy-mm-dd-hh*:*mm*:*ss*

/	For date fields: *mm*/*dd*/*yyyy or yyyy*/*mm*/*dd* For timestamp fields: *mm*/*dd*/*yyyy*/*hh*:*mm*:*ss* or *yyyy*/*mm*/*dd*/*hh*:*mm*:*ss*
#	*yyyy* Month *dd* for date fields
1	Commas, no sign, 0.0
2	Commas, no sign, blank
3	No commas, no sign, 0.00
4	No commas, no sign, blank
5	Explicit, CR/DR
6	Commas, '*' as suffix
7	Commas, c as prefix, '-'
8	Commas, 'c' as prefix, '-'
9	Edit word for date field, '-' as separator
A	Commas, CR, 0.00
B	Commas, CR, blank
C	No commas, CR, 0.00
D	No commas, CR, blank
J	Commas, '-', 0.00
K	Commas, '-', blank
L	No commas, '-', 0.00
M	, '-', blank
P	Edit word for phone number
R	No commas, no sign, 0.00
S	Edit word for U.S. social security number
T	Edit word for time fields
W	Edit word for long dates *dd*/*mm*/*ccyy*
Y	Edit word for date fields
Z	Zero suppression only

Using Field Edit Codes

Edit codes are used to alter or customize panel or report design. You use edit codes to punctuate data fields for panel entry, panel display and report output. CA 2E ships a number of edit codes that you can use. The shipped file is YEDTCDERFP, residing in library Y2SY.

If you are using ENPTUI (enhanced NPT user interface) you can choose to mask input edit codes.

Starting in Release 6.0, the CA 2E DDS generator follows these rules:

- When resolving the edit word, if 'field length+1' results in an edit word with a zero in the left-most portion of the edit word, an EDTWRD with a leading zero will be generated.
- If the above situation is not encountered, the edit word is generated without any changes.

This satisfies Date, Time, and Social Security number requirements where all leading zeros are needed. However, existing limitation in the DDS generation of edit words still remain, including:

- Certain fields with long lengths and many decimal places fail at compile time when combined with certain EDTWRD definitions.
- If the first character in an EDTWRD is a format character, as in a phone number, that character will not display for NPT DDS. This limitation can be corrected by allowing for an integer to the left of the left-most format character and using an edit mask.

See the chapter "Modifying Device Designs" in *Building Applications* for more information on ENPTUI input edit code masking.

To obtain a list of edit codes, type a **?** in any of the Edit code fields on the Edit Field Details panel. Depending on your selection (screen input, screen output, or report output), the Display Edit Codes panel shows a list of codes available for the selected field.

You can change the shipped edit code masks directly in the shipped file.

Description and Usage of Field Types

Code (CDE)

The following table describes the default characteristics of the CDE Field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxCD	-	Y	N
System data type	Alpha	Y	N	N
External length	6	N	Y	N
Internal length	as external	-	-	-
Decimal places	-	-	-	-
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Code	Y	Y	Y
Keyboard shift	XANWIDM	N	Y	N
Allow lowercase	N	Y	Y	N
Mandatory fill	N	Y	Y	N
Valid system name	N	Y	Y	N
Mod 10/11 check	-	-	-	-
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	Required	Y	N	Y
Edit codes: Input	-	-	-	-
Output	-	-	-	-
Report	-	-	-	-

The CDE type is used for fields that represent codes. Fields of CDE type are alphanumeric and are typically keys to a file. The valid set of values for a CDE field is controlled by their existence as the primary key to a database file.

Examples of CDE fields include:

- Product code
- Currency code
- Warehouse location code

Eight-digit Date

The following table describes the default characteristics of the D8# Field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxDX	-	Y	N
System data type	Packed	N	N	N
External length	6.0	N	N	N
Internal length	8.0	N	N	N
Decimal places	0	N	N	N
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Date	Y	Y	Y
Keyboard shift	DYN	N	N	N
Allow lowercase	-	-	-	-
Mandatory fill	N	Y	Y	N
Valid system name	-	-	-	-
Mod 10/11 check	N	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	Y	Y	N	Y
Edit codes: Input	4	Y	Y	Y
Output	/	Y	Y	Y
Report	/	Y	Y	-

The D8# type is one of three field types used for fields that represent dates. For compatibility with standards set by the International Standards Organization (ISO), it is recommended that you use DT# for your date fields. You can use the *MOVE built-in function to convert among date fields of type DTE, DT#, and D8#.

Note: *MOVE converts the three date types to a field of type NBR differently.

Internal Format

D8# dates are stored internally in YYYYMMDD format in systems generated by CA 2E.

The following table shows the internal format for April 5 for various years.

Date	Internal D8# Format
April 5, 2106	21060405
April 5, 2006	20060405
April 5, 1906	19060405
April 5, 1806	18060405

The valid date range is January 1, 1801 to December 31, 2199. When no date is entered, the internal representation is zero (0).

The internal format ensures that

- An historical view can be obtained. In this format dates can be ordered into ascending or descending order using the database.
- The stored internal format is independent of the displayed external format, namely, independent of the date format used in a particular country.

External Format

The external format for D8# fields and the valid range for entering dates depends on the input edit code you select for the field.

- If the edit code has a 4-digit-year format (*YYYY*), the range of dates you can enter is the same as the internal format range, namely, January 1, 1801 to December 31, 2199.
- If the edit code has a 2-digit year format (*YY*), the range of dates you can enter is restricted to a 'floating' hundred years (00-99) starting from the year CA 2E specified by the YCUTOFF model value. The cut-off year can be any year from within the range of 1900-1999 and its current value is retrieved at run time. The shipped default is 1940. In this case a year greater than or equal to 40 is assumed to be in the 20th century; a year less than 40 is assumed to be in the 21st century.

The external format for D8# fields for both input and output also depends on the setting of the Date Generation Validation (YDATGEN) and Date Format (YDATFMT) model values. It can be **MDY*, **DMY*, or **YMD*. Note that to enter, display, or print date fields with a four-digit-year external format such as, *MM*/*DD*/*YYYY*, you need to change the appropriate edit code to either / or –.

For more information on four-digit years and edit codes, refer to the table in the description of the DT# field type in this chapter.

See the chapter "Modifying Action Diagrams" in *Building Applications* for more information on *MOVE and the date built-in functions.

Validation

D8# fields are automatically validated by CA 2E. Dates are automatically translated from external to internal format and vice-versa. Dates are validated to be in the external format when entered on a panel and converted to internal format when written to a file.

The date is not converted if its day, month, or year is out-of-range. Instead, the message "Invalid date" is issued and the date is redisplayed in reverse image according to the output edit code.

Examples of D8# fields include:

- Date of birth
- Order date
- Creation date

ISO Date

The following table describes default characteristics of the DT# field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxDZ	-	Y	N
System data type	Alpha	N	N	N
External length	6.0	N	N	N
Internal length	10	N	N	N
Decimal places	N	N	N	N
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Date	Y	Y	Y
Keyboard shift	DYN	N	N	N
Allow lowercase	-	-	-	-
Mandatory fill	-	Y	Y	N
Valid system name	-	-	-	-
Mod 10/11 check	-	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	Y	Y	N	Y
Edit codes: Input	4	Y	Y	Y
Output	Y	Y	Y	Y
Report	Y	Y	Y	-

The DT# type is one of three field types used for fields that represent dates. You can use the *MOVE built-in function to convert among date fields of type DTE, DT#, and D8#.

Note: *MOVE converts the three date types to a field of type NBR differently.

See the chapter "Modifying Action Diagrams" in *Building Applications* for more information on this conversion and date built-in functions.

For compatibility with standards set by the International Standards Organization (ISO), it is recommended that you use DT# for your date fields. Since the DT# field type meets ISO standards, date fields of this type are interpreted correctly for SQL and Query Manager.

Internal Format

ISO dates are stored internally in *YYYY-MM-DD* format in systems generated by CA 2E. The following table shows the internal representation for April 5 for various years.

Date	Internal DT# Format
April 5, 2106	2106-04-05
April 5, 2006	2006-04-05
April 5, 1906	1906-04-05
April 5, 1806	1806-04-05

The valid date range is January 1, 1801 to December 31, 2199. If no date is entered, the internal representation is 0001-01-01.

The internal format ensures that

- ISO dates can be ordered into ascending or descending order using the database.
- The stored internal format is independent of the displayed external format, namely, independent of the date format used in a particular country.

External Format

The external format for DT# fields and the valid range for entering dates depends on the input edit code you select for the field.

- If the edit code has a 4-digit-year format (*YYYY*), the range of dates you can enter is the same as the internal format range, namely, January 1, 1801 to December 31, 2199.
- If the edit code has a 2-digit year format (*YY*), the range of dates you can enter is restricted to a 'floating' hundred years (00-99) starting from the year specified by the YCUTOFF model value. The cut-off year can be any year from within the range of 1900-1999 and its current value is retrieved at run time. The shipped default is 1940. In this case a year greater than or equal to 40 is assumed to be in the 20th century; a year less than 40 is assumed to be in the 21st century.

The external format for DT# fields for both input and output also depends on the setting of the Date Generation Validation (YDATGEN) and Date Format (YDATFMT) model values. It can be *MDY, *DMY, or *YMD. Note

Note: To enter, display, or print date fields with a four-digit-year external format, such as *MM*/*DD*/*YYYY*, you need to change the edit codes to either / or – .

The following table shows how the edit codes and the settings of the YDATGEN and YDATFMT model values affect the way that dates are displayed and printed.

Note: This table also applies to user-defined 8-digit date fields that have field type 'DT8'.

YDATGEN	YDATFMT (Run time)	Input Edit Code	Output Edit Codes	Date Displayed or Printed As:
*MDY	—	4	Y	10/27/95
		/	/	10/27/1995
		–	–	10-27-1995
*DMY	—	4	Y	27/10/95
		/	/	27/10/1995
		–	–	27-10-1995
*YMD	—	4	Y	95/10/27
		/	/	1995/10/27
		–	–	1995-10-27
*VRY	*MDY	4	Y	10/27/95
		/	/	10/27/1995
		–	–	10-27-1995

YDATGEN	YDATFMT (Run time)	Input Edit Code	Output Edit Codes	Date Displayed or Printed As:
*VRY	*DMY	4	Y	27/10/95
		/	/	27/10/1995
		–	–	27-10-1995
*VRY	*YMD	4	Y	95/10/27
		/	/	19/95/1027 (2)
		–	–	19-95-1027 (2)

Note: Due to limitations within DDS, you cannot produce this result at run time with YDATGEN set to *VRY and YDATFMT set to *YMD. To display or print digit years in *YMD format you need to set YDATGEN to *YMD.

Validation

DT# fields are automatically validated by . ISO dates are automatically translated from external to internal format and vice-versa. Dates are validated to be in the external format when entered on a panel, and converted to the internal format before being written to a file.

The date is not converted if its day, month, or year is out-of-range. Instead, the message "Invalid date" is issued and the date is redisplayed in reverse image according to the output edit code.

CA 2E generates ISO dates as the i OS Date type with DATFMT(*ISO) and assimilates i OS Date fields as type DT#.

Examples of DT# fields include:

- Date of birth
- Order date
- Creation date

Date (DTE)

The following table describes the default characteristics of the DTE field

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxDT	-	Y	N
System data type	Packed	N	N	N
External length	6.0	N	N	N
Internal length	7.0	N	N	N
Decimal places	0	N	N	N
LHS text (Column headings)	Field name	N	Y	Y
	Date	Y	Y	Y
RHS text				
Keyboard shift	DYN	N	N	N
Allow lowercase	-	-	-	-
Mandatory fill	N	Y	Y	N
Valid system name	-	-	-	-
Mod 10/11 check	N	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	Blank	Y	N	Y
Edit codes: Input	4	Y	Y	Y
Output	Y	Y	Y	Y
Report	Y	Y	Y	-

The DTE type is one of three field types used for fields that represent dates. For compatibility with standards set by the International Standards Organization (ISO), it is recommended that you use DT# for your date fields. Use the *MOVE built-in function to convert among date fields of type DTE, DT#, and D8#.

Note: *MOVE converts the three date types to a field of type NBR differently.

Internal Format

DTE dates are stored on file by systems generated by CA 2E as follows.

- DTE dates greater than or equal to 1900 are stored on file in CYYMMDD format.
- DTE dates earlier than 1900 are stored on file as a negative value.

The following table shows the internal format for April 5 for various years.

Date	Internal DTE Format
April 5, 2106	2060405
April 5, 2006	1060405
April 5, 1906	60405
April 5, 1806	939595–

The valid date range is January 1, 1801 to December 31, 2199. When no date is entered, the internal representation is zero (0).

The internal format ensures that

- An historical view can be obtained. In this format dates can be ordered into ascending or descending order using the database.
- The stored internal format is independent of the displayed external format, namely, independent of the date format used in a particular country.

External Format

The external format for DTE fields and the valid range for entering dates depends on the input edit code you select for the field.

- If the edit code has a 4-digit-year format (*YYYY*), the range of dates you can enter is the same as the internal format range, namely, January 1, 1801 to December 31, 2199.
- If the edit code has a 2-digit year format (*YY*), the range of dates you can enter is restricted to a 'floating' hundred years (00-99) starting from the year specified by the YCUTOFF model value. The cut-off year can be any year from within the range of 1900-1999 and its current value is retrieved at run time. The shipped default is 1940. In this case a year greater than or equal to 40 is assumed to be in the 20th century; a year less than 40 is assumed to be in the 21st century.

The external format for DTE fields for both input and output also depends on the setting of the Date Generation Validation (YDATGEN) and Date Format (YDATFMT) model values. It can be *MDY, *DMY, or *YMD. Note that to enter, display, or print date fields with a four-digit-year external format such as, *MM*/*DD*/*YYYY*, you need to change the appropriate edit code to either / or –.

For more information on four-digit years and edit codes, refer to the table in the description of the DT# field type in this chapter.

See the chapter "Modifying Action Diagrams" in *Building Applications* for more information on *MOVE and the date built-in functions.

Validation

DTE fields are automatically validated by CA 2E. Dates are automatically translated from external to internal format and vice-versa. Dates are validated to be in the external format when entered on a panel and converted to internal format when written to a file.

A date cannot be converted if its day, month, or year portion has an out-of-range value. If you enter an invalid date, the message, "Invalid date" is issued and the date is redisplayed in reverse image according to the output edit code:

- If the edit code has a four-digit-year format (*YYYY*), the invalid day and month portions are reproduced as they were entered. An invalid year is replaced by the special value 9999 to indicate that the entered year is out of range.
- If the edit code has a two-digit-year format (*YY*), all date portions are reproduced with no change.

Examples of DTE fields include:

- Date of birth
- Order date
- Creation date

Ideographic Character Text

The following table describes the default characteristics of the IGC field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxIG	-	Y	N
System data type	Alpha	N	N	N
External length	20	-	Y	-
Internal length	as external	Y	-	N
Decimal places	-	-	-	-
LHS text (Column headings)	Field name	N	Y	Y
RHS text	IGC Text	Y	Y	Y
Keyboard shift	O	N	Y	N
Allow lowercase	-	N	-	-
Mandatory fill	N	Y	Y	N
Valid system name	N	Y	-	-
Mod 10/11 check	-	-	-	-

Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	N	Y	N	Y
Edit codes: Input	-	-	-	-
Output	-	-	-	-
Report	-	-	-	-

The IGC type is used for fields that contain both ideographic and alphanumeric data. Ideographic data consists of Japanese, Korean or Chinese characters. The fields have the default keyboard shift O.

The source generated for files containing IGC fields automatically contains the necessary keywords, such as IGCCNV. Examples of IGC fields include:

- Customer name
- Customer address

For more information on keyboard shifts and ideographic enhancements, refer to *IBM i DDS Reference*, Appendix I.

Narrative Text

The following table describes the default characteristics of the NAR field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxNA	-	Y	N
System data type	Alpha	N	N	N
External length	30	Y	Y	N
Internal length	as external	-	-	-
Decimal places	-	-	-	-
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Text	Y	Y	Y
Keyboard shift	XANWIDM	N	Y	N
Allow lowercase	Y	Y	Y	N
Mandatory fill	N	Y	Y	N
Valid system name	N	Y	Y	N
Mod 10/11 check	-	-	-	-

Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	N	Y	N	Y
Edit codes: Input	-	-	-	-
Output	-	-	-	-
Report	-	-	-	-
Multi-Line Entry	N	N	Y	Y

The NAR type is used for fields that represent narrative text. NAR field attribute type should be used in contrast with the TXT field type, which specifies a basic descriptive title for an (entity) file, such as Company name or Product name. The NAR attribute type can be regarded as a catch-all to define data fields not covered by any other field types.

Examples of NAR fields include:

- Order comments
- Address lines

Number (NBR)

The following table contains Default characteristics of the NBR Field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxNB	-	Y	N
System data type	Packed	N	N	N
External length	5.0	Y	Y	N
Internal length	as external	-	-	-
Decimal places	0	Y	Y	N
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Number	Y	Y	Y
Keyboard shift	NSYDI	N	Y	N
Allow lowercase	-	-	-	-
Mandatory fill	N	Y	Y	N
Valid system name	-	-	-	-
Mod 10/11 check	N	Y	-	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-

Field exit option	Zero	Y	N	Y
Edit codes: Input	4	Y	Y	Y
Output	3	Y	Y	Y
Report	3	Y	Y	-

The NBR type is used for fields that contain purely numeric data values. By default, NBR fields are for integers.

Numeric fields with characteristics such as VAL, QTY, PCT, PRC, should use these field types since they provide a more precise specification. The NBR type can be regarded as a catch-all for numeric data fields that are not covered by other types.

Examples of NBR fields include:

- Number of customers
- Number of records in file

Note: When prompting on numeric fields, the **?** cannot be used. The prompt function key (F4) allows prompting. The YCUAPMT model value must be set to Y to enable F4 for prompting.

See the YCHGMDLVAL command in CA 2E *Command Reference Guide* for more information on changing model values.

Percentage (PCT)

The following table contains the Default characteristics of the PCT Field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxPC	-	Y	N
System data type	Packed	N	N	N
External length	5.2	Y	Y	N
Internal length	as external	-	-	-
Decimal places	2	Y	Y	N
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Percent	Y	Y	Y

Keyboard shift	NSIDY	N	Y	N
Allow lowercase	-	-	-	-
Mandatory fill	N	Y	Y	N
Valid system name	-	-	-	-
Mod 10/11 check	N	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	Blank	Y	N	Y
Edit codes: Input	4	Y	Y	Y
Output	3	Y	Y	Y
Report	3	Y	Y	-

The PCT type is used for fields that represent a percentage or a part of a whole expressed in hundredths.

Examples of PCT fields include:

- Percentage purity
- Percentage market share
- Percentage usage
- Profit margin
- Market index

Price (PRC)

The following table contains the Default characteristics of the PRC Field

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxPR	-	Y	N
System data type	Packed	N	N	N
External length	7.2	Y	Y	N
Internal length	as external	-	-	-
Decimal places	2	Y	Y	N
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Price	Y	Y	Y

Keyboard shift	NSIYD	N	Y	N
Allow lowercase	-	-	-	-
Mandatory fill	N	Y	Y	N
Valid system name	-	-	-	-
Mod 10/11 check	N	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	Blank	Y	N	Y
Edit codes: Input	4	Y	Y	Y
Output	3	Y	Y	Y
Report	3	Y	Y	-

The PRC type is used for fields that represent a price such as a monetary rate or value per unit.

Price fields are typically used to represent a value per unit. PRC field type should be used in contrast with these field types:

- Pure numeric value (NBR): if the number does not have a standard characteristic, such as Line number, a pure numeric type should be used.
- Numeric fields with other standard characteristics (VAL, PCT): ensure that the field is a price and not a value.

Examples of PRC fields include:

- Retail price
- Manufacturing price
- Discount price
- Customs tariff

Quantity (QTY)

The following table contains the Default characteristics of the QTY Field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxQT	-	Y	N
System data type	Packed	N	N	N
External length	5.0	Y	Y	N
Internal length	as external	-	-	-
Decimal places	N	Y	Y	N
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Quantity	Y	Y	Y
Keyboard shift	YNSID	N	Y	N
Allow lowercase	-	-	-	-
Mandatory fill	N	Y	Y	N
Valid system name	-	-	-	-
Mod 10/11 check	N	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	Blank	Y	N	Y
Edit codes: Input	4	Y	Y	Y
Output	3	Y	Y	Y
Report	3	Y	Y	-

The QTY type is used for fields that represent quantities; that is, numbers of a given characteristic in standard units. A clear specification of characteristics facilitates verification and use of the data model.

The units measured by a quantity depend on your business. For property it may be square feet, for garment retailing it may be meters of fabric, for pharmaceuticals it may be milligrams. The criteria to be considered in assigning the attribute QTY to a field is not what the actual quantity is but whether it is a quantity.

QTY field type should be used in contrast with the following other field types:

- Pure numeric value (NBR): if the number does not have a standard characteristic such as Line number, a pure numeric type should be used.
- Numeric fields with other standard characteristics (VAL, PCT): you should ensure that the field is a quantity and not a value.

Examples of QTY fields include:

- Yards of fabric (m)
- Floor space (sq.m)
- Gross tonnage (T)

Reference Field (REF)

The REF field type is used to specify that the definition of a field is based on the definition of another field. The name of the referenced field must be specified in the definition. The referencing field is given the same attributes as the referenced field but has a different field name.

The text, check and default conditions, and generation name are unique for each field. You can override these attributes.

The field length, data type, usage, and edit codes are shared. A REF field can inherit narrative (help text) from the referenced field.

You can specify different check and default conditions for the referenced and referencing fields.

For example, an existing STS field called State has all of the state abbreviations listed as conditions. If you need two states for an Order, two fields can be created: Ship To State and Bill to State. Both of these new fields would be REF fields, referencing the field State. They can now share the same conditions of the field to which they refer, State.

Defining a group of similar fields as REF fields, based on one particular field, ensures that all the fields in the group belong to the same domain as defined in CA 2E.

CA 2E carries out domain checking to ensure that only a field of the right type and size is passed as a parameter to a function.

REF fields do not share the same true domain. They do, however, share the same set of conditions. Where CA 2E requires a field to be in the same domain, the field must be the parent field or a REF field referencing the parent.

Examples of REF fields include:

- Delivery quantity (REF Order quantity)
- Manager code (REF Employee code)
- Sub-area code (REF Area code)
- Array element 2 (REF Array element 1)

Defining Function Fields as REF Fields

If you define a function field with a usage type of SUM, CNT, MIN, or MAX as REF field, the input parameter to the field function will be the referenced field.

See the chapter "Defining Functions" in *Building Applications* for more information on specific function fields.

See the chapter "Modifying Device Designs" in *Building Applications* for more information on adding, changing, or modifying function fields.

Surrogate (SGT)

The following table contains the default characteristics of the SGT field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxSG	-	Y	N
System data type	Packed	N	N	N
External length	7.0	Y	Y	N
Internal length	as external	-	-	-
Decimal places	0	Y	Y	N
	N			
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Quantity	Y	Y	Y
Keyboard shift	N	N	Y	N
Allow lowercase	-	-	-	-
Mandatory fill	N	Y	Y	N
Valid system name	-	-	-	-
Mod 10/11 check	N	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	N	Y	N	Y
Edit codes: Input	3	Y	Y	Y
Output	3	Y	Y	Y
Report	4	Y	Y	-

The SGT field type should be used to designate a field that is a system-assigned key. Surrogate is a numeric value.

Examples of SGT fields include:

- System key
- Alternate key

Status (STS)

The following table contains the default characteristics of the STS Field:

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxST	-	Y	N
System data type	Alpha	N	N	N
External length	1	Y	Y	N
Internal length	1	Y	Y	N
Decimal places	-	-	-	-
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Values list	Y	Y	Y
Keyboard shift	XANWID	N	Y	N
Allow lowercase	N	Y	Y	N
Mandatory fill	N	Y	Y	N
Valid system name	N	Y	Y	N
Mod10/11 check	-	-	-	-
Check condition	*NONE	N	Y	Y
Translate values	N	N	Y	N
Field exit option	N	Y	N	Y
Edit codes: Input	-	-	-	-
Output	-	-	-	-
Report	-	-	-	-

The STS field type is used for fields that are indicators or flags. Status fields can take a limited number of discrete values, each of which has a meaning assigned to it.

The values for a status field and their meanings are specified through the use of specific field conditions allowed for status fields. If a check condition is specified for a status field, it can only take the values specified by that condition.

For value mapped status fields, you can specify that a status field has a different internal length from its external length; the value field is then mapped between the two values.

For more information on valued mapped status fields, refer to this topic, Field Type Default Characteristics, the Translate values option.

A call to an inquiry program is automatically generated for status fields that appear as input capable fields in CA 2E standard function panel displays. The inquiry shows the allowed values for a status field if you press F4 (prompt) or enter **?** in the selection area of the panel.

Only conditions of type VAL or LST can be attached to status fields.

See the section Using Conditions for more information.

The following examples show fields of STS type and the conditions that can be attached to them:

- Order status (Ordered, Paid, Held, and Canceled)
- Product Status (Active, Not active)
- Quality (Passed, Failed, Under test)
- Allow refund (Yes, No).

ISO Time (TM#)

The following table contains the default characteristics of the TM# field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxTA	-	Y	N
System data type	Alpha	N	N	N
External length	6.0	N	N	N
Internal length	8	-	-	-
Decimal places	N	N	N	N
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Time	Y	Y	Y

Keyboard shift	DYN	N	N	N
Allow lowercase	-	-	-	-
Mandatory fill	-	Y	Y	N
Valid system name	-	-	-	-
Mod10/11 check	-	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	Y	Y	N	Y
Edit codes: Input	T	Y	Y	Y
Output	T	Y	Y	Y
Report	T	Y	Y	-

The TM# type is one of two field types used for fields that represent times. For compatibility with standards set by the International Standards Organization (ISO), it is recommended that you use TM# for your time fields. Since the TM# field type meets ISO standards, time fields of this type are interpreted correctly for SQL and Query Manager.

CA 2E automatically generates code to validate TM# fields.

ISO time is stored internally in HH.MM.SS format and externally as HHMMSS.

Note: A value of 00.00.00 on the physical file represents zero, not a valid time.

CA 2E generates ISO times as the i OS Time type with TIMFMT(*ISO) and assimilates i OS Time fields as type TM#.

Examples of TM# fields include:

- Time of birth
- Time of order
- Time of creation

Time (TME)

The following table contains the default characteristics of the TME field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxTM	-	Y	N
System data type	Packed	N	N	N
External length	6.0	N	N	N
Internal length	as external	-	-	-
Decimal places	0	N	N	N
LHS text (Column headings)	Field name	N	Y	Y
RHS text	HH:MM:SS	Y	Y	Y
Keyboard shift	NYID	N	N	N
Allow lowercase	-	-	-	-
Mandatory fill	-	Y	Y	N
Valid system name	-	-	-	-
Mod10/11 check	-	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	Blank	Y	N	Y
Edit codes: Input	T	Y	Y	Y
Output	T	Y	Y	Y
Report	T	Y	Y	-

The TME type is one of two field types used for fields that represent times. For compatibility with standards set by the International Standards Organization (ISO), it is recommended that you use TM# for your time fields. You can use the *MOVE built-in function to convert between date fields of type TME and TM#.

CA 2E automatically generates code to validate TME fields. TME times are always stored on file in HHMMSS format.

Examples of TME fields include:

- Time of birth
- Time of order

ISO Timestamp (TS#)

The following table contains the default characteristics of TS# field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxTS	-	Y	N
System data type	Alpha	N	N	N
External length	18.0	N	N	N
Internal length	26	N	N	N
Decimal places	N	N	N	N
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Timestamp	Y	Y	Y
Keyboard shift	DYN	N	N	N
Allow lowercase	-	-	-	-
Mandatory fill	-	Y	Y	N
Valid system name	-	-	-	-
Mod10/11 check	-	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	blank	Y	N	Y
Edit codes: Input	4	Y	Y	Y
Output	/	Y	Y	Y
Report	/	Y	Y	-

The TS# type is used for fields that represent timestamps. Since the TS# field type meets standards set by the International Standards Organization (ISO), timestamp fields of this type are interpreted correctly for SQL and Query Manager.CA 2E automatically generates code to validate TS# field.

ISO timestamp is stored internally in YYYY-MM-DD-HH.MM.SS.NNNNNN format. External formats are:

External Format	Valid Edit Codes
MM-DD-YYYY-HH:*MM*:*SS* or	-
YYYY-MM-DD-HH:*MM*:*SS*	/
MM/*DD*/*YYYY*/*HH*:*MM*:*SS* or	T and Y
YYYY/*MM*/*DD*/*HH*:*MM*:*SS*	
MM/*DD*/*YY*/*HH*:*MM*:*SS*:*NNNNNN*	

Note: A value of 0001-01-01-00.00.00.000000 on the physical file represents zero, not a valid timestamp.

The external format for TS# fields for both input and output also depends on the setting of the Date Generation Validation (YDATGEN) and Date Format (YDATFMT) model values.

For more information on how edit codes and the settings of YDATGEN and YDATFMT affect the way in which timestamps are displayed and printed, refer to the table in the description of the DT# field type in this chapter.

You can use the *MOVE built-in function to convert between timestamp fields and time and date fields.

CA 2E generates ISO timestamp as the i OS Timestamp type and assimilates i OS Timestamp fields as type TS#.

Examples of TS# fields include:

- Process ending date and time
- Transaction audit date and time
- Date and time of creation

Text (TXT)

The following table contains the default characteristics of the TXT field

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxTX	-	Y	N
System data type	Alpha	N	N	N
External length	25	Y	Y	N
Internal length	as external	-	-	-
Decimal places	-	-	-	-
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Text	Y	Y	Y
Keyboard shift	ANWID	N	Y	N
Allow lowercase	Y	Y	Y	N
Mandatory fill	N	Y	Y	N
Valid system name	N	Y	Y	N
Mod10/11 check	-	-	-	-
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	N	Y	N	Y
Edit codes: Input	-	-	-	-
Output	-	-	-	-
Report	-	-	-	-
Multi-Line Entry	N	N	Y	Y

The TXT type is used for fields that represent text description. It can be used to define alphanumeric fields that are not appropriate for type CDE or STS.

The TXT field should be used to provide a title for an object, such as on inquiries. The use of the TXT attribute should be contrasted with the narrative text (NAR) type, which is used for additional descriptive comments.

Examples of TXT fields include:

- Customer name
- Currency name
- Member name
- Country name

Value (VAL)

The following table contains the default characteristics of the VAL field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxVA	-	Y	N
System data type	Packed	N	N	N
External length	11.2	Y	Y	N
Internal length	as external	-	-	-
Decimal places	2	Y	Y	N
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Monetary value	Y	Y	Y
Keyboard shift	NSYID	N	Y	N
Allow lowercase	-	-	-	-
Mandatory fill	N	Y	Y	N
Valid system name	-	-	-	-
Mod10/11 check	N	Y	Y	N
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	Blank	Y	N	Y
Edit codes: Input	4	Y	Y	Y
Output	C	Y	Y	Y
Report	C	Y	Y	-

The VAL type is used for fields that represent a monetary value, such as an amount in units of a particular currency. The VAL type should be used in contrast with the following field types:

- Pure numeric value (NBR): if the number does not have a standard characteristic such as Line number, a pure numeric type should be used.
- Numeric fields with other standard characteristics (QTY, PCT): ensure that the field is a value as opposed to a quantity.

Examples of VAL fields include:

- Value of order ($)
- Value of stock holding ($)
- Customer credit limit ($)

Valid System name

The following table contains the default characteristics of the VNM field.

Field Type Attribute	Shipped Default Value	Default Override	Field Details Override	Device Field Override
Implementation name	xxVN	-	Y	N
System data type	Alpha	N	N	N
External length	10	Y	Y	N
Internal length	as external	-	-	-
Decimal places	-	-	-	-
LHS text (Column headings)	Field name	N	Y	Y
RHS text	Name	Y	Y	Y
Keyboard shift	XAI	N	Y	N
Allow lowercase	N	Y	Y	N
Mandatory fill	N	Y	Y	N
Valid system name	Y	Y	Y	N
Mod10/11 check	-	-	-	-
Check condition	*NONE	N	Y	Y
Translate values	-	-	-	-
Field exit option	N	Y	N	Y
Edit codes: Input	-	-	-	-
Output	-	-	-	-
Report	-	-	-	-

The VNM type is used for fields that represent system entities including objects, formats, and field names. A valid system name must

- Begin with a letter, $, # or @
- Contain no more than ten characters
- Contain only characters, digits, or the characters $, #, @, and underscore
- Contain no embedded blanks
- For i OS, must be uppercase

This type is used for defining fields that must conform to the operating system's naming convention. For DDS, it is implemented by using the CHECK(VN) keyword.

Examples of VNM fields include:

- Member name
- Job name
- User name

Using Function Fields

Function fields are used to hold data requiring calculated values such as order quantity, location space, or population size. Function fields are given specific usage types. The usage types determine the allowable values of a function field.

Function fields specify work or calculation fields that can be used in CA 2E functions. Function fields can be placed on device designs and defined to represent special derivatives like summations, maximum and minimum values.

Function Field Usages

Type	Description	Data Type	Example
DRV	User defined logic	Any	Value = Price + Quantity
USR	User-defined field	Any	Control total
CNT	Count of records	Numeric	Number of lines on order
MAX	Maximum value	Numeric	Largest item on order
MIN	Minimum value	Numeric	Smallest item on order
SUM	Sum of fields	Numeric	Order total

Usage types of function fields are listed and described in the pages that follow.

See the chapter "Defining Functions" in *Building Applications* for more information.

Count (CNT)

The CNT usage type is given to those function fields that contain a count of the number of records containing the field (to be counted) over a series of records.

A function field of usage type CNT has two parameters:

- Result parameter, which is the derived field itself. This must be placed on a total format of any function that calls the CNT function.
- Input parameter attribute to be counted. This must be present on the detail format of the calling function.

The series of records are defined by the standard function within which the field is defined. For instance, in a Print File function, the CNT field could calculate the number of times a field occurs in the detail lines of the report, to be displayed in the next total format.

Fields of type CNT must be numeric. If the function field is defined as a REF field, based on another field,CA 2E assumes that the CNT field is a count of the number of records containing the based-on field. This method of definition cannot be used if the field to be counted is not itself a numeric field.

Examples of CNT fields include:

- Number of employees on employee file
- Number of customers in company file for a company
- Number of items within warehouse

Derived (DRV)

The DRV usage type is given to function fields that perform a user-defined calculation specified by an action diagram. The field can then be used in any function where the calculation is required. This method of definition cannot be used if the field to be counted is not itself a numeric field.

A DRV function field has one output parameter: the derived field itself. Input parameters may be specified for the function.

You can edit the action diagram and specify the parameters for the derived field.

Examples of DRV fields include:

- Order line value
- Discounted order value

Maximum (MAX)

The MAX usage type is given to function fields that contain the highest value found for a field over a series of records.

A function field of usage type MAX has two parameters:

- **Result parameter**—The derived field itself. This must be placed on a total format of any function, which calls the MAX function.
- **Input parameter**—The field for which the highest value is to be determined. This must be present on the detail format of the calling function.

The series of records are defined by the standard function within which the field is defined. For instance, within an Edit Transaction function, a MAX field defined on the header could be used to calculate the maximum value of a field on the subfile record.

Fields of type MAX must always be numeric. If the function field is defined as a REF field, based on another numeric field, assumes that the based-on field is the field whose highest value is to be calculated. This method of definition cannot be used if the field to be calculated is not itself a numeric field.

Examples of MAX fields include:

- Largest order item : maximum of order quantity
- Biggest warehouse location : maximum of location size
- Largest town : maximum of town size
- Highest line number : maximum of line number

Minimum (MIN)

The MIN usage type is given to function fields that contain the lowest value found for a field over a series of records.

A function field of usage type MIN has two parameters:

- **Result parameter**—The derived field itself. This must be placed on a header format of any function, which calls the MIN function.
- **Input parameter**—The field for which the lowest value is to be determined. This must be present on the details format of the calling function.

The series of records are defined by the standard function within which the function field is defined.

For instance, within an Edit Transaction function, a MIN field defined on the header could be used to calculate the minimum value of a field on the subfile record.

Fields of usage type MIN must always be numeric. If the function field is defined as a REF field, based on another numeric field,CA 2E assumes that the based-on field is the field whose lowest value is to be calculated. This method of definition cannot be used if the field to be calculated is not itself a numeric field.

Examples of MIN fields include:

- Smallest order item : minimum of order quantity
- Smallest warehouse location : minimum of location size
- Smallest town : minimum of town size

Summation (SUM)

The SUM usage type is given to function fields, which contain the sum of the values found for another field over a series of records.

A function field of usage type SUM has two parameters:

- **Result parameter**—The field containing the result of the summation.

  This field must be placed on a total format of any function that calls the SUM function.

- **Input parameter**—The field for which the sum is to be calculated. This must be present on the detail format of the calling function.

The series of records are defined by the standard function within which the field is defined. For instance, in a Print File function, the SUM field could calculate the sum of the values in a field from the detail lines of the report, which is to be displayed in the next total format.

Function fields of type SUM must always be numeric. If the function field is defined as a REF field, based on another numeric field,CA 2E assumes that it is the sum of the values in the based-on field that is to be calculated. This method of definition cannot be used if the field to be calculated is not itself a numeric field.

Examples of SUM fields include:

- Total order value : sum of order line value
- Total warehouse space : sum of location space
- Total population size : sum of area population size

User-Defined (USR)

The USR usage type is given to any field that you wish to add to a function device design. You can make such a field input capable if you wish.

It is your responsibility to initialize and process the USR fields in the action diagram.CA 2E performs basic field checking, such as date validation.

Examples of USR fields include:

- Order total check value
- Command request string
- Next menu option

Using Conditions

A CA 2E condition both specifies the values or set of values that a field may take and indicates what those values mean.

Conditions are used to

- Validate the entry of data
- Specify the select/omit criteria for access paths
- Specify processing conditions in action diagrams
- Condition the appearance of fields on function device formats
- Specify function parameter values when calling functions in action diagrams
- Specify default field values for adding records to a database

Properties of Conditions

Each condition has a name, a type, and an associated value. All the conditions associated with a single field must be unique.

Condition Types

A condition type specifies the type of validation rule it imposes. CA 2E has four types of conditions, divided into two categories: those that are used with status fields and those used with non-status fields.

The condition types allowed for status fields are:

- VAL (Value)
- LST (Value List)

The condition types allowed for non-status fields are:

- CMP (Compare)
- RNG (Range)

This table lists the valid condition types and the field types to which they can be attached. The use of the different types of field conditions is described in the sections following the table.

Condition Type	Description	Example	Field Type
CMP	Compare using an operator	Greater than 5	All others
RNG	Valid range between two values	0–20	All others
VAL	Value	A	STS
LST	List of value conditions	Held, Paid, Unpaid	STS

See the chapter "Modifying Action Diagrams" in *Building Applications* for more information on using conditions involving functions.

Status Field Conditions

The VAL (Value) and LST (Value List) are the two conditions allowed for use with status fields. You can use these condition types to:

- Specify single (VAL) and multiple values (LST)
- Specify value mapping (VAL)

See the chapter "Maintaining Your Data Model" for more information on specifying value mapping and converting conditions to values list for status fields.

Value (VAL) Condition

This condition type is used to specify single values that a status field may take.

Internal and External Values

You can specify two related values for a VAL condition:

- **Internal value**—The value held on the implemented database file to represent the condition
- **External value**—The value entered by and displayed to the user in functions

CA 2E automatically generates source to translate between the two values. The internal and external values may have different lengths. This value mapping facility may be used to facilitate translation into different national languages. Value mapping only takes place if a value is specified for the Translate condition (cnd) values field on the Edit Field Details panel and if a Check condition value is specified for the field.

If you add or modify values within that LST condition, you will need to recompile the functions that use the field. The validation check will then include your changes.

The following table contains examples of VAL conditions.

Condition	File Value	Translate Condition Value	Display / Input Value
Full-time Employee	F	N	F
Part-time Employee	P	N	P
Invoice status is 'Held'	H	Y	HLD
Invoice status is 'Paid'	P	Y	PAD
Invoice status is 'Delivered'	D	Y	DLV

If no Translate cnd values is specified, this condition is implemented using the VALUES keyword in the display file DDS.

If Translate cnd values is specified, this condition is implemented by HLL code specifying that the condition is to be checked against a CA 2E-created database file.

See the chapter "Maintaining Your Data Model" for more information on conditions and converting condition values.

List (LST) Condition

This condition type is used for conditions that specify a list of values that a status field may take. Each LST condition is made up of one or more VAL conditions.

When you specify this condition type, a special LST condition, *ALL values, is created as soon as a condition for the field is defined.

LST conditions have a special use in specifying field value checking in Display Device type functions. If you specify a Check condition for a field using the Edit Field Details panel or the Edit Screen Entry Details panel, CA 2E generates the necessary code to ensure that any value entered is a valid condition in the list for all interactive functions that use that field.

If you add or modify values within the LST condition, you will need to recompile the functions that use the field. The validation check will then include your changes.

Condition List Inquiries

In generated functions, you can display the list of values available for a status field in either of two ways: enter **?** in the field, or place the cursor on the field and press F4. The F4 to prompt only works if the YCUAPMT model value is set to Y (Yes).

For more information on using the YCHGMDLVAL command to change the value for YCUAPMT model value, refer to the *CA 2E Command Reference Guide*.

Examples of LST Conditions

The following five value conditions are attached to an Invoice Status field:

- LST Condition—All Values

Value Condition	File Value
Invoice not yet due	U
Invoice due	D
Invoice paid	P
Invoice held	H
Invoice canceled	C

You may then define and create two different LST conditions; for example, Invoices Outstanding and Completed Invoices, using a combination of the above VAL conditions:

- LST Condition—Outstanding Invoices

Value Condition	File Value
Invoice not yet due	U
Invoice due	D
Invoice held	H

- LST Condition—Completed Invoices

Value Condition	File Value
Invoice paid	P
Invoice canceled	C

Non-Status Field Conditions

The conditions that can be attached to non-status fields include the CMP (Compare) and RNG (Range).

Compare (CMP) Condition

This condition type is used for conditions that specify values that a non-status field may take, defined in terms of a fixed value and an operator.

Valid Operators

- EQ—Equal to
- NE—Not equal to
- GT—Greater than
- LT—Less than
- GE—Greater than or equal to
- LE—Less than or equal to

Examples of CMP Condition:

Field name	Condition name	Operation	Value
Order quantity	Greater than 10	GT (>)	10
Credit limit	Less than $100.00	LT (<)	100
License date	Less than expire date (Enter as *YYMMDD*)	LT (<)	991203

This condition is implemented using the COMP keyword in the display file DDS.

Range (RNG) Condition

This condition type is used for conditions that specify a range of values that a non-status field may take, in terms of two fixed values between which the field value must lie, end points included.

Examples of RNG Conditions:

Field name	Condition Name	From	To
Stock quantity	Between 10 and 100	10	100
Transaction value	GT -250 and LT 250	250-	250

This condition is implemented using the RANGE keyword in the display DDS file.

See the chapter "Maintaining Your Data Model" for more information on using conditions.

Using Relations

This topic provides conceptual information and a full description of CA 2E relations. It also explains and includes examples of how different types of relations are used within your model.

CA 2E Relations

A CA 2E relation expresses an association between two files or between a file and a field. Relations constitute the fundamental links in a data model. They enable you to make assertions about the meaning of the connections within your data.

CA 2E uses basic English verbs to describe relations as shown in these examples:

```
Customer   Known by              Customer code

Customer  Has            Customer name

Customer  Refers to      Salesperson
```

The Refers to relation indicates that a relationship exists between the two files (Customer-Salesperson); Known by and Has indicate the relationship between the file and the field (Customer-Customer code, or Customer-Customer name).

Several relations can be specified for a single file. CA 2E automatically resolves the relations of a file into the fields that are needed to implement that file. The fields that result from resolving a relation are called file entries.

Relation Types

There are eight types of relations. An understanding of the purpose of these relation types is central to understanding CA 2E.

- Defined as
- Extended by
- Has
- Includes
- Known by
- Owned by
- Qualified by
- Refers to

Relation Usage Groups

Depending on how they operate within a data model, CA 2E relations are grouped under three different usage groups:

- Definition relations, which declare files to exist
- Key relations, which define the keys that identify a file by reference either to the keys of another CA 2E file or to a field
- Attribute relations, which declare the non-key fields that are present in a file by reference either to another file or to a field

The use of specific CA 2E relations allows for the physical arrangement, selection, and retrieval of information based on the arrangement of key fields in the database. The relations you choose determine which fields are to appear on which file, whether a field is a key field or foreign key field, and whether certain fields can be shared between files.

You must use the appropriate CA 2E relation types to describe a file-to-file or file-to-field relation respectively.

CA 2E Relations

Usage Type	Relation	Used For
Definition	Defined as	File-to-file relationship
Key	Owned by	File-to-file relationship
	Known by	File-to-field relationship
	Qualified by	File-to-field relationship

Attribute	Refers to	File-to-file relationship
	Includes	File-to-file relationship
	Has	File-to-field relationship
Other	Extended by	File-to-file relationship

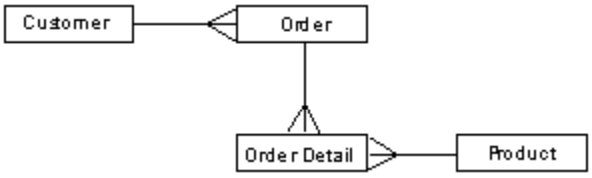
Example of Relations Used in a Data Model

The following examples focus on the CA 2E relation types: Owned by, Known by, Qualified by, Has, and Refers to.

Example 1: Simple Sales Ledger

You have a number of customers who order your products. Each order can involve a number of different products but can be issued to only one customer. Customer and Product are identified by Customer code and Product code respectively; an order is identified by an Order number that is unique within the business. Each Order is made up of an Order header and a variable number of Order detail lines, each of which is for a particular quantity of a particular Product.

The situation can be represented diagrammatically as follows:



We could model this situation by using the CA 2E relation statements described below.

- A Customer could be described as follows:

```
FIL       Customer          REF   Known by          FLD   Customer code   CDE

FIL  Customer          REF   Has       FLD   Customer name   TXT
```

The key of the Customer file is the Customer code.

- A Product could be described as follows:

```
FIL        Product   REF   Known by          FLD   Product code    CDE

FIL  Product   REF   Has       FLD   Product name    TXT

FIL  Product   REF   Has       FLD   Product size    NBR
```

- An Order could be described as follows:

```
FIL Order  CPT  Known by        1 FLD          Order code      CDE
```

```
FIL Order  CPT Has     2 FLD          Order date      DT#

FIL Order  CPT       Has     3 FLD          Order status    STS

FIL Order  CPT       Refers to       4 FIL          Customer                REF
```

```
FIL Order Detail  CPT   Owned by       1 FIL          Order           CPT

FIL Order Detail  CPT       Known by       2 FLD          Order line no   NBR

FIL Order Detail  CPT       Has            3 FLD          Order quantity  QTY

FIL Order Detail  CPT       Refers to      4 FIL          Product         REF
```

The previous relations result in the following entries in the files:

	Product
K	Product code
	Product name
	Product size

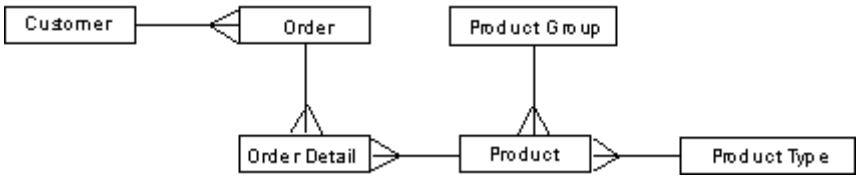
	Customer
K	Customer code
	Customer name

	Order
K	Order code
	Order date
	Order status
	Customer code

	Order Detail
K	Order code
K	Order line no
	Order quantity
	Product code

Example 2: A More Complicated Product Structure

Your products have a more complicated structure than first imagined. Each Product belongs to a Product group that serves as part of the product identifier. Each Product also has a Product type that is not part of the product identifier, but has some extra information associated with it. We can refine the model to reflect this situation by adding a Product group and a Product type file.



A Product group could be described as follows:

```
FIL Product group  REF    Known by        FLD     Product group code      CDE
FIL Product group  REF       Has     FLD     Product group name      TXT
```

A Product type could be described as follows:

```
FIL Product type  REF   Known by        FLD     Product type code       CDE
FIL Product type  REF        Has     FLD     Product type name       TXT
FIL Product type  REF        Has     FLD     Pack size               QTY
FIL Product type  REF        Has     FLD     Freight charge          VAL
```

The definition of a Product could then be amended by adding an Owned by relation to associate each Product with Product Group and a Refers to relation to associate a Product type with each Product.

```
FIL Product       REF     Owned by        Product group   FIL     REF
FIL Product          REF     Known by        Product code    FLD     CDE
FIL Product          REF     Has     Product name    FLD     REF
FIL Product          REF     Has     Product size    FLD     QTY
FIL Product          REF     Refers to       Product type    FIL     REF
```

We now have entries on two new files, the Product group and the Product type. A new entry for the Product group code has been added wherever the Product file was referenced by other CA 2E files, in this case, on the Order details:

Product Group	
K	Product group code
	Product group name

Product Type	
K	Product type code
	Product type name
	Pack size
	Freight charge

	Product
K	Product group code
K	Product code
	Product name
	Product size
	Product type code

	Order Detail
K	Order code
K	Order line no
	Order quantity
	Product group code
	Product code

Specifying Relations

This topic provides information for using the eight CA 2E relation types and includes examples that illustrate and explain how each individual type is used.

You specify a relation through the Edit Database Relations panel. Relations are specified as relation statements that have the following format:

Subject	Relation	Object
OBJ	REL	OBJ

Subject OBJ is the name of a file, REL is the name of a relation type, and Object OBJ is the name of a file or field.

File-to-file Relationships

The most important aspects of your data model are described by the relations that connect the files within the model.

To describe file-to-file relations you use:

- Owned by
- Extended by
- Refers to
- Includes

File-to-field Relationships

File-to-field relations are used to explicitly state that a field is to be present on a file.

You can use these three types of CA 2E relations to describe a file-to-field relationship:

- Known by
- Qualified by
- Has

Describing and Using CA 2E Relations

This section specifies how each relation type is used and provides examples.

Defined as Relation

The Defined as relation declares that a file exists. For example, to say a file named "Product" exists:

Product	Defined as	Product

A Defined as relation is present for each file.CA 2E implicitly creates the Defined as relation if the definition of the file was not done using the Defined as relation in the first place.

The Defined as relation does not cause any field entries to be added to a file.

Examples of Using Defined as Relation

Example 1: Defining a Single Entity

Let us say that a rose is a rose, declared with the following relations:

```
FIL        Rose    REF    Known by       FLD    Any other name  CDE
FIL  Rose    REF     Has     FLD     Petals          TXT
FIL  Rose    REF     Has     FLD     Thorns          NBR
FIL  Rose    REF     Has     FLD     Rose Type       STS
```

This will automatically result in the following additional relation:

```
FIL        Rose    REF     Defined as      FIL     Rose    REF
```

Example 2: Defining Several Entities (Top-down)

The Defined as statement constitutes the most basic way of declaring an entity to exist. If you are creating a new model working top down, you may first declare all the entities that you think will be required to define your model. Example:

```
FIL Company   REF       Defined as      FIL     Company         REF
FIL Division  REF    Defined as      FIL     Division                REF
FIL Product   REF    Defined as      FIL     Product         REF
FIL Customer  REF    Defined as      FIL     Customer                REF
```

There must be a Defined as relation for every CA 2E file. If you make a reference to a non-existing file, a Defined as statement is created automatically for the file.

In practice, you seldom need to enter a Defined as statement explicitly, unless you choose to define the entities in your model before using them in any other relation.

Displaying Defined as Relations

Defined as relations are not normally shown on the Edit Database Relations panel. You can display Defined as relations by entering DFN or ALL in the relation level (Rel lvl) field on the positioner line at the top of the Edit Database Relations panel.

Deleting Defined as Relations

A Defined as relation cannot be deleted until all references to the file are also deleted.

Owned by Relation

The Owned by relation denotes a parent-child relationship. The primary key(s) of the owning file become part of the primary key of the owned by file. For instance, if Order Detail is Owned by Order, the key of Order, Order code, is the high order key of Order Detail.

The file of an Owned by relation must be of type REF or CPT. All the key fields of the owning file are incorporated as high order keys in the owned file.

A file can have more than one Owned by relation. A file can have only Owned by relations to define its keys. It is not necessary to have other key relations, such as Known by or Qualified by.

The Owned by relation allows you access to any of the fields on the owning file. If a field is accessed on the owning file, it creates a virtual field on the owned file. Virtual field values can be used but cannot be updated.

An Owned by relation can have its description clarified by using the For text extension.

See the section Using for Text and Sharing with Relations for more information on using text and Sharing with Owned by relations.

Examples of Using Owned by Relations

Example 1: Orders within Company

The Owned by relation may be used to specify the high order key of a file. Let us say that you operate a multi-company sales ledger and that all orders are within company.

A Company could be defined as follows:

```
FIL    Company        REF    Known by       FLD    Company code    CDE
FIL  Company        REF    Has    FLD    Company name    TXT
```

An Order could be defined as follows:

```
FIL    Order  REF    Owned by       FIL    Company        REF
FIL  Order  REF    Known by       FLD    Order code     CDE
FIL  Order  REF    Has    FLD    Order date     DT#
```

This specifies that the key of the Company file is the high order key of the Order file, which results in the following entries:

Company	
K	Company code
	Company name

Order	
K	Company code
K	Order code
	Order date

Example 2: Orders within Company within Country

Owned by relations may be used to construct a hierarchy. In the example given above, the Owned by statement asserts that the keys of Company are the high order keys of Order. If you later decide that a Company is only unique within Country, then adding Country to the Company file with an Owned by relation will automatically add it to the Order file. The presence of the relation stating Order is Owned by Company causes the automatic addition.

A Country could be defined as follows:

```
FIL    Country        REF     Known by       FLD     Company code    CDE
FIL  Country        REF     Has     FLD     Company name    TXT
```

Company could then be redefined as follows:

```
FIL       Order   REF     Owned by       FIL     Company         REF
FIL  Order   REF     Known by       FLD     Company code    CDE
FIL  Order   REF     Has     FLD     Company name    TXT
```

The definition of Order requires no change:

```
FIL       Order   REF     Owned by       FIL     Company         REF
FIL  Order   REF     Known by       FLD     Order code      CDE
FIL  Order   REF     Has     FLD     Order date      DT#
```

This results in the following entries, where you can see the Country code has been introduced automatically onto the Order file:

Country File	
K	Country code
	Country name

Company File	
K	Country code
K	Company code
	Company name

Order File	
K	Country code
K	Company code
K	Order code
	Order date

Known by Relation

The Known by relation specifies that a field is the key field, or one of the key fields of a file. This means that records in the file can be uniquely identified by the value of this field together with the values of any other key fields.

The field specified as the object of a Known by relation is added as a key field entry to the file containing the relation.

Examples of Using Known by Relation

Example 1: A Single Known by Relation - Company

Suppose you wish to identify companies by a company code.

A Company could be defined as follows:

```
FIL  Company        REF    Known by       FLD    Company code    CDE
FIL  Company        REF    Has      FLD   Company name    TXT
```

This results in the following entries:

```
         Company file
K        Company code
         Company name
```

Example 2: Multiple Known by Relations - Manager

There may be more than one Known by relation on a file. For example:

```
FIL   Event   REF     Known by        FLD     Date            DT#

FIL  Event   REF     Known by        FLD     Time            TM#

FIL  Event   REF     Has     FLD     Location                TXT
```

Note that the presence of more than one Known by relation on a file may indicate that an entity has been omitted from the model. For instance, consider the following relations to define a Manager:

```
FIL     Manager         REF     Known by        FLD     Manager type            STS

FIL  Manager         REF     Known by        FLD     Manager code            CDE

FIL  Manager         REF     Has     FLD     Manager name            TXT

FIL  Manager         REF     Has     FLD     Salary                  VAL
```

Neither Manager name nor Manager salary are properties of Manager type, which suggests that Manager type should be an entity in its own right, and that Manager should be Owned by Manager type.

Qualified by Relation

The Qualified by relation can be used to qualify a file identifier by one or more variable factors such as the date, the time, or a sequence number.

The Qualified by relation would typically be used for entities that represent a continuum of values. An example may be prices or currency rates that come into effect on a given date and prevail for a while. The identification of such entities may be qualified by a date.

Another common usage would be to describe step functions, such as volume discount breaks or tax ranges, which similarly come into effect at a certain threshold and prevail until the next threshold is reached.

Qualified by relations are further specified to tell whether the record retrieval is to be *PREVIOUS, to retrieve the nearest record less than or equal to the search value, or *NEXT, to retrieve the nearest record greater than or equal to the search value.

Note: These values are mutually exclusive; you cannot specify both on the same file.

To specify these values, enter + in the subfile selector of the Qualified by relation, then press F5.

Qualified by relations are similar to Known by relations: they are resolved by adding the named field as a key field to the file containing the relation. The Qualified by relation, however, has a special property: a reference to a file containing a Qualified by relation may be redirected.

The field usage required for the field of a Qualified by relation needs to be ATR and not CDE.

See the chapter "Creating/Defining Your Data Model" for more information on redirection.

Examples of Using Qualified by Relations

Example 1: A Qualified File - Product Prices

Your company has a number of products:

```
FIL    Product REF     Known by        FLD     Product code            CDE

FIL  Product         REF     Has     FLD     Product description     TXT
```

Your product prices change from time to time. You may then describe a Product price file as follows:

```
FIL      Product price   REF     Owned by        FIL     Product         CDE

FIL  Product price   REF     Qualified by    FLD     Effective date  DT#

FIL  Product price   REF     Has             FLD     Price           PRC
```

Thus for each change of Product price you would have a separate record, a state of affairs represented by the following entries:

Example 2: Using a Qualified File - Product Prices

If you now wish to use the Product price in an Order detail file, you could define an Order file as follows:

```
FIL    Order   REF     Known by        FLD     Order number            CDE

FIL  Order   REF     Has             FLD     Order date              DT#

FIL  Order   REF     Has             FLD     Order status            STS

FIL  Order   REF     Refers to       FIL     Customer                        REF
```

And, define an Order detail file as follows:

```
FIL      Order detail    REF     Owned by        FLD     Order           CPT

FIL  Order detail    REF     Known by        FLD     Order line no   NBR

FIL  Order detail    REF     Has             FLD     Order quantity  QTY

FIL  Order detail    REF     Refers to       FIL     Product price   REF
```

This would result in the following entries:

	Order detail file
K	Order number
K	Order line no
	Order quantity
	Product code
	Effective date

The fields on the file are the same as if you had used a Known by relation instead of a Qualified by relation for the Effective date. However, additional processing logic is created for the Qualified by relation.

The difference in using a Known by instead of a Qualified by relation is that you will have a code generated that refers to the correct product price. The code is based not on an exact value match of the effective date but on the closest previous value of the effective date.

Extended by Relation

The Extended by relation declares a file to be an extension of another file. The relation records an association that is not expressed by any other relation, and is, in particular, a one-to-one or one-to-none association between the identifiers of two files.

When it is used with an existing Owned by relation, the Extended by relation has no effect on the entries of the file being extended; it merely makes the fields from the extended file available for selection as virtual fields on the file being extended.

A virtual field is logically present in a view of a file, though it physically resides in another file.

See the section Adding Virtual Fields to File to File Relations for more information.

A one-to-none relation denotes a relation where an instance exists in one file and the corresponding instance does not exist in another file. For example, the Product file could be extended by the export details file. Some products may be exported and some may not. This would mean that a product record may or may not have an associated record in the export detail file.

Note: It is not recommended that you use the Extended by relation unless the extending file is Owned by the extended file, as unpredictable results may occur during source code generation.

Example of Using Extended by Relations

Your basic Customer information consists of the following fields:

```
FIL       Customer       REF     Known by       FLD     Customer code          CDE
FIL  Customer        REF     Has            FLD     Customer name          TXT
FIL  Customer        REF     Extended by    FIL     Customer detail        REF
```

You may enter additional details about Customers in another file called Customer detail:

```
FIL  Customer detail REF     Owned by       FIL     Customer               CDE
FIL  Customer detail REF     Has            FLD     Credit limit    VAL
FIL  Customer detail REF     Has            FLD     Managing Director      TXT
```

This results in the following entries:

Customer file	
K	Customer code
	Customer name

Customer detail	
K	Customer code
	Credit limit
	Managing Director

If you wish to create a function that brings both Customer and Customer detail together, you need to build an access path that contains fields from both files. If you were doing this on the Customer detail file, this would not present a problem since you could specify virtual fields on the Owned by relation

```
FIL    Customer detail REF     Owned by        FIL     Customer               CDE

     VRT     Customer name    TXT

FIL  Customer detail REF     Has             FLD     Credit limit    VAL

FIL  Customer detail REF     Has             FLD     Managing Director       TXT
```

However, if you wish to attach your function to the Customer file, you would not be able to obtain the customer details unless you had an Extended by relation. Using the Extended by relation, you can specify virtual fields as follows:

```
FIL       Customer detail REF     Known by        FLD     Customer code          CDE

FIL  Customer detail REF     Has             FLD     Customer name   TXT

FIL  Customer detail REF     Extended by     FIL     Customer detail REF

                                             VRT     Credit limit    VAL

                                             VRT     Managing Director       TXT
```

Each Customer may have only one Customer detail record: there is a one-to-one correspondence between files.

There are two implementation reasons why you may consider using the Extended by relation rather than simply including the data from the extended file in the basic file:

- To save space. If some data fields are only present on a minority of records, then it may be desirable to place the rarely used fields into an Extended by file.
- To avoid recompilation of an existing system. If you wish to add fields to an existing file that is already used by a large number of programs, you could avoid level check problems by placing the extra fields in another file owned by the original file. An Extended by relation would make the new file details available from the based-on file.

  Note: An Extended by relation effectively constrains an Owned by relation, which is normally one-to-many, to be a one-to-one relationship. It is only appropriate to use the Extended by relation for cases where a one-to-one or one-to-none relationship holds.

Refers to Relation

The Refers to relation specifies that a file references another file. A Refers to relation is resolved by including the identifiers (keys) of the referenced file into the referring file as foreign key fields.

The Refers to relation allows access to any of the fields on the referred to file from the referring file. If a field is accessed on the referred to file, it creates a virtual field on the referring file. Virtual field values can be used but cannot be updated.

A virtual field is logically present in a view of a file, though it physically resides in another file.

See the chapter "Maintaining Your Data Model" for more information on virtual fields.

A Refers to relation can have a For text extension to further clarify its description.

For more information on using For text and Sharing with Refers to relations, see the section Using For text and Sharing with Relations.

The Refers to relation can be contrasted with the Includes relation, which includes all fields from the referenced file, and with the Owned by relation, which is resolved into key entries on the owned file.

Note: Up to 60 Refers to relations can be placed on a file.

Example of Using Refers to Relations

Where Order detail refers to Product, Product may be defined as follows:

```
FIL      Product REF     Known by        FLD     Product code            CDE
FIL  Product REF     Has             FLD     Product name                    TXT
```

The Product could then be referenced elsewhere, for instance by an Order detail file:

```
FIL     Order detail    REF     Owned by        FIL     Order           CPT
FIL  Order detail    REF     Known by        FLD     Order line no   NBR
FIL  Order detail    REF     Has             FLD     Order quantity  QTY
FIL  Order detail    REF     Refers to       FIL     Product         REF
```

This results in the Product code being added to the Order detail file as a foreign key field:

	Product file		Order detail file
K	Product code	K	Order code
	Product name	K	Order line no
			Order quantity
			Product code

Has Relation

The Has relation declares a field to be present in a file as an attribute. Each field declared as a subject of a Has relation for a file is included in the file as a non-key field.

Example of Using Has Relations

A company is defined as follows:

```
FIL  Company REF     Known by       FLD     Company code    CDE
FIL  Company REF     Has     1      FLD     Company name            TXT
FIL  Company REF     Has     2      FLD     Creation date   DT#
FIL  Company REF     Has     3      FLD     Profit last year        VAL
FIL  Company REF     Has     4      FLD     No of employees NBR
```

These relationships will result in the following entries:

	Company file
K	Company code
	Company name
	Creation date
	Profit last year
	No of employees

For each CA 2E file, all the file-to-field relations must be unique. The same field cannot be declared twice in the same file. A field can be declared in two different files. Although it can be declared with two different usages, this is not recommended. For example:

```
FIL      Product         REF     Known by        FLD     Product code            CDE
FIL  Order Detail    REF     Has             FLD     Product code    CDE
```

Using a field as an attribute in one file and as an identifier in another file usually indicates that a relation is missing from your model.

Includes Relation

The Includes relation states that a file is to include fields that have already been declared as being present in a structure file. The Includes relation allows the use of a group of fields or "a data structure," in several different files.

Specifying an Includes relation causes all of the fields in the included file to be present as non-key fields in the including file.

See the section Using Files for more information on structure files.

Examples of Using Includes Relations

You have an Audit stamp structure file that is made up of three components:

```
FIL   Audit stamp          STR    Has          FLD    Date          DT#
FIL  Audit stamp           STR    Has          FLD    Time          TM#
FIL  Audit stamp           STR    Has          FLD    User ID       CDE
```

You may wish to refer to the Audit stamp in a number of files.

For instance, a product file:

```
FIL       Product REF     Known by       FLD     Product code          CDE
FIL  Product REF     Has           FLD     Product name                TXT
FIL  Product REF     Has           FLD     Product quality       QTY
FIL  Product REF     Includes      FIL     Audit stamp                 STR
```

For instance, an order file:

```
FIL      Order   REF     Known by       FLD     Order number          CDE
FIL  Order   REF     Has           FLD     Order status                STS
FIL  Order   REF     Includes      FIL     Audit stamp                 STR
```

This would result in the following entries:

Product	Order

K	Product code
	Produce name
	Product quantity
	Date
	Time
	User ID

K	Order number
	Order status
	Date
	Time
	User ID

Relation Sequencing

The relations that describe a file are resolved into entries in the order they are specified on that file.

The following table contains the default sequence order for CA 2E relations.

Usage group	Relation
Definition	Defined as
Key	Owned by
	Known by
	Qualified by
Attribute	Extended by
	Refers to
	Has
	Includes

You can change this default sequence. You can control the order in which CA 2E resolves relations by using the sequence field on the relation statements.

See the chapter "Creating/Defining Your Data Model" for more information on changing the sequence order of relations.

Using For Text and Sharing with Relations

A CA 2E relation is specified in the form of a relation statement, consisting of the referencing file, a referenced file or field, and the relationship between them. The relationships can be further clarified using a For text clause.

For Text

You can add a For text clause to a relation statement to further clarify the description of a relationship. Owned by and Refers to are the only relations that can use For text.

To extend a relation, type a **+** in the selection field beside the Owned by or Refers to relation and press F5.

The For text helps document the meaning of a relation. The For text is also used to identify file entries that may be duplicates of existing file entries based on a previously defined relationship.

If a given file refers to another file more than once, the For parameter can be used to distinguish between each reference.

CA 2E files cannot contain duplicate fields. To prevent duplicate fields from being added to a file from the resolution of the Owned by and Refers to relations, CA 2E uses the following procedure:

- If the new entry arises from a relation that has For text, CA 2E uses the For text and the entry name to define the new field.

  Example**:** If Company code is the entry that arises from a relation that has Invoice in its For text, the new field will be Invoice Company code, with a REF field type, referencing Company code.

- CA 2E then checks the field dictionary to determine whether Invoice Company code exists. If it does, CA 2E uses the field. If it does not, creates the Invoice Company code as a new field to be added to the file, referencing the existing field (Company code).

- If the new entry arises from a relation without For text,CA 2E uses the entry name and a surrogate number to define the new field. The surrogate number is added as part of the entry name.

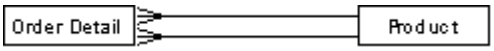
Example**:** If Company code is the new entry, the new field may be Company code 25642.

The new field has a field type of REF, referencing the existing field (Company code).

You may override this processing and modify the names of fields by using the Display Referenced Field Details panel.

Examples of Using For Text

You have two entities defined, Customer and Order. The Order is placed by one Customer but can be paid for by a different Customer. The Order needs to have two references to the Customer entity to define the two Customers, one for ordering and one for invoicing.



To clarify which Refers to relation is for invoicing Customer and which is for ordering Customer, use For text.

```
FIL   Customer        REF     Known by        FLD     Customer code   CDE

FIL  Customer        REF     Has             FLD     Customer name   TXT

FIL  Order   REF     Known by        FLD     Order code      CDE

FIL  Order   REF     Refers to       FIL     Customer                        TXT


       For: Ordering                   Sharing: *ALL


FIL  Order   REF     Refers to       FIL     Customer                        REF


              For: Invoicing                 Sharing: *ALL
```

The resolved entries for the two entities will be:

	Customer
K	Customer code
	Customer name
	Order
K	Order code
	Customer code
	Invoicing Customer code

Note that when a file A Refers to a file B more than once, the For text is applied only to the second and subsequent Refers to relations. To change the first relation, type R (replace field) against the first relation on the Edit Field Entries panel and define a new referenced field on the Display Referenced Field Details panel with the For text appended; in our example, Ordering Customer code. The resolved entries for the Order file would then be:

	Order
K	Order code
	Ordering Customer code
	Invoicing Customer code

Sharing

Sharing means you want to share or choose a specific instance or key value in the chain of relationships. This is not just an implementation specification; it relates business requirements as well.

Sharing only takes place if the referenced entity has more than one key field; only the high order keys may be shared. The low order key will always require a separate entry.

Take the following model for example:

```
Customer is Owned by Company

Order Refers to Customer For Ordering Customer

Order Refers to Customer For Invoicing Customer
```

An Order refers to the Customer twice, first for Ordering Customer and then for Invoicing Customer. This requires two entries in Order of Customer Code, one for Ordering and one for Invoicing.

For the Company Code entry, there is a choice. If the two customers (ordering and invoicing) must be customers of the *same* company, then, to ensure this, the Company Code is shared for the two Refers to relations. That means there would be only one Company Code entry in the Order file. If the two customers can be customers of different companies, then the Company Code is not shared for the two Refers to relations. That is when two Company Code entries in the Order file are needed, one for ordering and one for invoicing.

The Owned by and Refers to relations may imply that a key field should be added to a file when that field already exists on the file because it was resolved from a preceding relation, causing duplicate entries.

You can control whether separate entries are created for a field in a file-to-file relation by checking the value specified for the Sharing parameter on the relation statement. For example:

- If *NONE is specified for the Sharing parameter, a separate entry is added to the file for all the fields.
- If *ALL is specified for the Sharing parameter, the high order keys may be shared. The low order key will always have a separate entry.
- If a file name is specified for the Sharing parameter, the entry is shared if it is present in the specified file and in both the owned and owning files. A separate entry is added if the file name or *ALL is not specified in either of these files.

You can use the Default Sharing Type (YSHRDFT) model value to set the default sharing to *NONE or *ALL.

Example of Sharing

A Product Bill of Materials requires two Owned by relations from the Product file to the Assembly file. One relation represents the Parent Product; the other represents the Component Product. The Product file is owned by the Division file, which is owned by the Company file. You use Sharing to specify that the Components of a Product must be from the same Company as the resulting or Parent Product but that the components can be from different divisions.

The Sharing text specifies that the two Owned by relations share the Company file record.

See the chapter "Creating/Defining Your Data Model" for more information on sharing entries and redirection.

Use of For Text for a Parts Assembly

This example is intended to show the use of For text to distinguish between different entries of the same field on a single file. This example includes the use of Sharing.

A Company is defined as follows:

```
FIL   Company REF     Known by        FLD     Customer code   CDE
FIL  Company REF     Has             FLD     Customer name   TXT
```

```
FIL    Division        REF     Owned by        FIL     Company         REF
FIL  Division        REF     Known by        FLD     Division code   CDE
FIL  Division        REF     Has             FLD     Division name   TXT
```

And each Company is split into divisions, thus a Division is defined as follows:

```
FIL  Division        REF     Owned by        FIL     Company         REF
FIL  Division        REF     Known by        FLD     Division code   CDE
FIL  Division        REF     Has             FLD     Division name   TXT
```

Let us now introduce the products handled by the company and say that each Division produces different products or parts. This means that a Part is to be defined as follows:

```
FIL     Part    REF     Owned by        FIL     Division                REF
FIL  Part    REF     Known by        FLD     Part code                       CDE
FIL  Part    REF     Has             FLD     Part name                       TXT
```

The above relations result in the following entries. Note that Part has both Company code and Division code in its key:

Company file	
K	Company code
	Company name

Division file	
K	Company code
K	Division code
	Division name

Part file
K Company code
K Division code
K Part code
Part name

The basic description of an Assembly includes two separate references to a Part, one as Resulting Part and one as Component Part. You can distinguish between the two by use of the For text:

```
FIL        Assembly        REF     Owned by        FIL     Part            REF

       For: Resulting                  Sharing: *ALL

FIL  Assembly        REF     Owned by        FIL     Part                  REF

       For: Component                  Sharing: *ALL

FIL  Assembly        REF     Has             FLD     Assembly qty          QTY
```

After the first instance of the Part code field on the Part file, the For text will be prefixed to each additional instance to create a unique entry name.

As a further consideration, you need to decide whether the Resulting parts and the Component parts belong to the same Division and Company. Whether they do or not is indicated by the value specified for the Sharing field. As a default, sharing is assumed. In this instance, it will be assumed that the Company and Division for both Component and Resulting parts is the same. The Component part must be from the same Company and Division as the resulting part. This means any fields that would be duplicated by the resolution of both Owned by relations will not actually be repeated.

The following entries would result:

Assembly file

K	Company code
K	Division code
K	Part code
K	Component Part code
	Assembly qty

Thus, the additional instances of Company code and Division code that may arise from the second Owned by relation have been suppressed.

If the Component belongs to different divisions and companies than the "Resulting" part, we would specify that there is no sharing of common keys; that is, specify a value of *NONE for the Sharing field. This causes any fields, whose presence would be duplicated on the generated file by the resolution of both of the Owned by relations, to be repeated with different names.

```
FIL       Assembly        REF     Owned by        FIL     Part            REF

       For: Resulting                 Sharing: *NONE

FIL Assembly        REF     Owned by        FIL     Part                    REF

       For: Component                 Sharing: *NONE

FIL Assembly        REF     Has             FLD     Assembly qty            QTY
```

This would result in the separate entries for the Resulting Company code, the Resulting Division code, and Resulting Part code, as follows:

Assembly file

K	Company code
K	Division code
K	Part code
K	Component Company code
K	Component Division code
K	Component Part code
	Assembly quantity

A third variation may be the case where components can be from divisions different than those of the resulting assembly but the resulting assembly and component must be for the same company. In this case you would specify that there is sharing only of Company. This causes the duplicate reference to the Company code to be dropped.

```
FIL    Assembly        REF     Owned by        FIL     Part            REF

       For: Resulting                  Sharing: Company

FIL      Assembly       REF     Owned by       FIL     Part                REF

       For: Component                  Sharing: Company

FIL      Assembly       REF     Has            FIL     Assembly qty        QTY
```

This would result in the following entries:

	Assembly file
K	Company code
K	Division code
K	Part code
K	Component Division code
K	Component Part code
	Assembly qty

Adding Virtual Fields to File-to-file Relations

The fact that a relation exists between two files means that, given a record from one file, it is possible to look for a corresponding record on the other file in order to obtain related data items.CA 2E allows you to specify which data items are to be obtained through a file-to-file relation by allowing you to specify virtual fields. A *virtual field* is a field that is present logically in a view of a file although it physically resides in another file.

Virtual fields can be specified on the following relations:

- Owned by
- Refers to
- Extended by

```
FIL       Order line          REF    Refers to      FIL    Product       REFT
                                                    VRT    Product name  XT
```

where:

Refers to—Represents a File to file relation

Product—Represents a CA 2E file containing field

Product name—Represents a virtual field name

Virtual fields can be defined only in one direction on the relation. In CA 2E, the file that can contain virtual fields is the file where the relation is defined. For example, Division is Owned by Company. Fields from the Company (owning file) can be virtualized to the Division (owned file). Fields from Division cannot be virtualized to Company. Division contains the virtual fields. Division is where the Owned by relation is defined.

A virtual field may itself be a virtual field on the referenced file.

You can specify virtual fields for relations using the Virtual Field Entries panel.

The example below shows how virtual fields are added to a relation:

- On the Order file we could have the Customer name as a virtual field:

```
FIL   Order   CPT    Known by       1 FLD   Order code      REF
FIL   Order   CPT    Has            2 FLD   Order date            CDE
FIL   Order   CPT    Has            3 FLD   Order status          TXT
FIL   Order   CPT    Refers to      4 FIL   Customer
```

```
                                        VRT   Customer name
```

- On the Order Detail file we could have both the Order information, including the Customer name and the Product name as virtual fields:

```
FIL  Order detail    CPT     Owned by        1 FLD   Order           REF
       VRT   Order date      DT#
       VRT   Order status    STS
       VRT   Customer code   CDE
       VRT   Customer name   TXT
FIL  Order detail    CPT     Known by        2 FLD   Order line no   CDE
FIL  Order detail    CPT     Has             3 FLD   Order quantity  QTY
FIL  Order detail    CPT     Refers to       4 FIL   Product REF
                                               VRT   Product name    TXT
```

- This would result in the following entries on the files:

	Order
K	Order code
V	Order date
	Order status
	Customer code
	Customer name

	Order detail
K	Order code
V	Order date
V	Order status
V	Customer code
V	Customer name
K	Order line no
	Order quantity
	Product code
V	Product name

See the chapter "Maintaining Your Data Model" for more information on how to add virtual fields to a relation.

Circularity

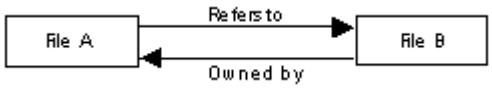
This topic addresses how circularity manifests itself and how to avoid virtualizing a field back onto the originating file.

If you can follow the path of relations from a file and end up returning to that file, you have an instance of *circularity* within the model. This does not necessarily mean that the series of relations is invalid, but that you must check the sequence of relations to ensure that the sequence will allow you to pass the virtual fields that you require. Circularity manifests itself in the disappearance and duplication of virtual fields.

Although the following three relations are acceptable, they can lead to circularity:

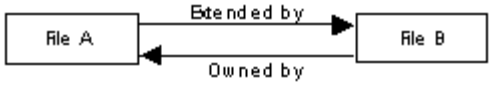
- Parent Refers to child

  A Refers to B, B Owned by A



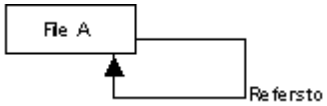
- Use of the Extended by relation

  A Extended by B, B Owned by A



- Self-referral

  Example: A Refers to A



Here is an example of a model containing these relations:

```
Account details         Owned by                Customer

      Account details Has             Account opened date



Customer                          Known by              Customer code

      Customer                          Extended by             Account details

      Customer                          Has                     Customer type
```

Virtualizing against the Owned by relation would allow you to declare Customer type as a virtual field on the Account details file, and you could declare Account opened date as a virtual field on the Customer file over the Extended by relation. The file entries would now look like this:

```
Account details         Customer code           Key

               Customer type           Virtual

        Account opened date     Attributes



      Customer                          Customer code           Key

                              Account opened date     Virtual

                                        Customer type           Attributes
```

If you then resynchronize the model, the virtual entry Account opened date no longer appears on the Customer file. During resynchronization, the relations are expanded into file entries. When expansion occurs, the Account details file relations will be expanded into entries before the Customer relations.

The expansion of relations would occur in this sequence:

1. Expand the relation, Account details Owned by Customer. The Customer file is still to be expanded, which you must do now before any virtual can be defined.
2. Expand the relation, Customer Known by Customer code. This results in the key entry Customer code on the Customer file.
3. Expand the relation, Customer Extended by Account details. The Account details file is not to be expanded, as the expansion started in step 1.

   No entries that exist on Account details can be virtual fields on the Customer file, since the relation, Account details Has Account opened date, has not been expanded.

4. Expand the relation, Customer Has Customer type. This results in the attribute entry, Customer type, on the Customer file.

   The expansion of Customer has finished, so it returns to Account details.

5. One entry exists on Customer that can be a virtual field on Account details: Customer type. This results in the field Customer type becoming a virtual entry on the Account details file.

6. Expand the relation, Account details Has Opened date. This results in the attribute entry, Account opened date, on the Account details file.

Expansion ends here. Two further steps are needed:

1. The Customer file must be expanded before the Account details file. You could do this by renaming the Customer file so that it comes before Account details alphabetically. This is not always satisfactory or easy. It would be better to add a file to the model that Refers to Customer, and itself has a name alphabetically lower than Account details. For example, a file name beginning with an asterisk (*) would serve this purpose. This file does not need to exist physically, because its sole purpose is to alter the expansion sequence within the model.

2. The Extended by relationship must be the last relation in the Customer file. You could do this simply by giving the Extended by relation a sequence number.

The model relations would now look like this:

```
  *Force Sequence         Refers to              Customer


       Account details Owned by               Customer

       Account details Has             Account opened date


      Customer                        Known by              Customer code

   Customer                        Has                   Customer type

               Customer                        Extended by            99 Account
details
```

The expansion of relations would happen in this sequence:

1. Expand the relation, *Force sequence Refers to Customer. The Customer needs to be expanded now before any virtual fields can be defined.

2. Expand the relation, Customer Known by Customer code. This results in the key entry, Customer code, on the Customer file.

3. Expand the relation, Customer Has Customer type. This results in the attribute entry, Customer type, on the Customer file.

4. Expand the relation, Customer Extended by Account details. The Account details file needs to be expanded now before any virtual fields can be defined.
5. Expand the relation, Account details Owned by Customer. The Customer file expansion started in step 1. One entry exists on Customer that can be virtualized on Account details: Customer type. This results in the virtual entry, Customer type, on the Account details file.
6. Expand the relation, Account details Has Account opened date. This results in the attribute entry, Account opened date, on the Account details file.

The expansion of Account details has finished, so it returns to *Force sequence. No virtuals will have been specified on the *Force sequence Refers to Customer relation. Expansion ends here.

If you find that virtuals have disappeared due to circularity in your model, you will also find that if you try to put them on again they will appear twice in the file entries. If this happens, repeatedly remove the virtuals until they do not appear in the file entries. Follow the steps above before adding them again.

Chapter 4: Creating/Defining Your Data Model

This chapter shows you:

- How to create a data model in CA 2E based on the conceptual model you developed earlier. You may have an ERD of your conceptual model ready to enter into CA 2E.
- How to work with file entries that are resolved from the CA 2E relations that you use to describe file relationships in your model.

See the chapter "Developing a Conceptual Model" for more information on how to produce an ERD.

This section contains the following topics:

Before You Begin

You should have created a design model, using the Create Model Library (YCRTMDLLIB) command, so that you can add information to it before using this module.

See the following:

- The chapter "Creating and Managing Your Model" in the *Administration Guide* for more information about preparing to use
- *Administration Guide* for information on how to set up model values
- *Building Access Paths* for information on how to build access paths
- *Building Applications* for information on how to build functions

Using CA 2E Model Management Facilities

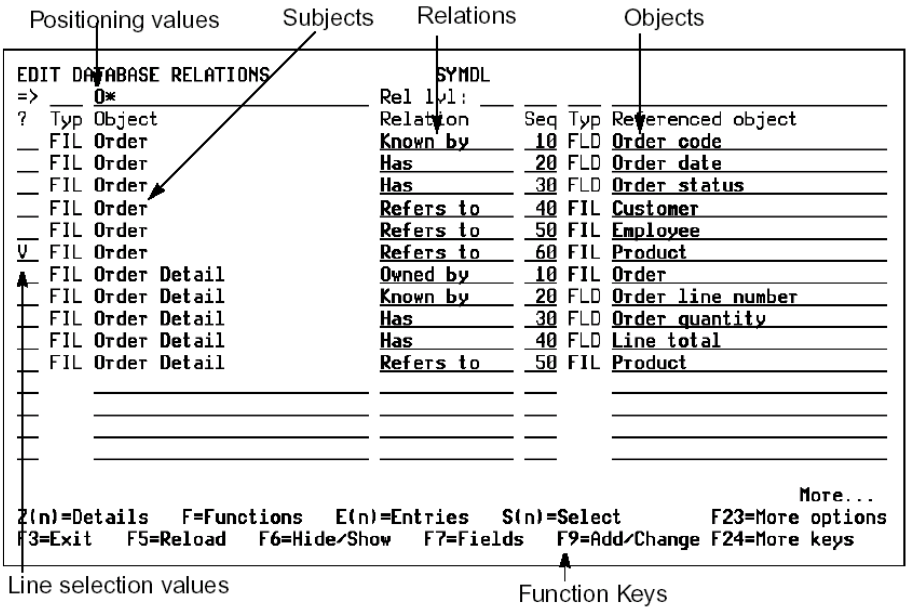
CA 2E provides facilities to help you manage your model, including the Edit Database Relations panel and the Edit Model Object List panel.

You can access a model as one of three different user types: designer, programmer, or user.

See the *Administration Guide* for more information on how to use CA 2E facilities. See the chapter "Using Your Development Environment" in the *Administration Guide* for more information on types of users.

Edit Database Relations Panel

The Edit Database Relations panel allows you to describe your data model to CA 2E. This is your starting point when creating a new data model. From here you can branch off to other areas in CA 2E.



Note: Although you need to use the Edit Database Relations panel to define files and relations, you can also use the Edit Model Object List panel to handle many of the other functions provided by the Edit Database Relations panel.

At the Edit Database Relations panel, you can edit your data model, access information, or navigate through CA 2E by using:

- Line selection values to perform model-related activities:
  - Add narrative text to describe your model at file, field, or relation level.

    N0, N, N1—narrative for model object

    N2—narrative for referenced object; field or file

    N3—narrative for Refers to with Sharing Relations. Available only on relations if sharing by an access path (not *None or *All)
  - Display all relations beginning with the "object" (S1) and relations that include the "referenced object" (S2).
  - Display all relations referring to the "object" or all relations beginning with the next file (T1, T2).
  - Virtualize (V).
  - Clarify a relation with For text (+ and F5)
  - Delete a relation (D).
  - Specify redirection (E0).
  - Go to the Edit File Entries panel (E); to the Edit Database Functions panel (F); to the Edit File Details panel (Z1 or Z); to the Edit Field Details panel (Z2).
- Function keys to define objects (F10), access online Help (Help key), the Data Dictionary (F7),CA 2E online map (F14) or CA 2E Display Services Menu (F17).
- Application areas to group your model files into specific categories under specific areas defined by a unique identification code. By specifying an application area code, you can choose to display or view only the part of the model (files) you wish, or use with documentation commands.

  Note: Alternatively, you can use model object lists to group together any combination of model object types; you are not restricted to grouping files. In addition, CA 2E provides many powerful tools and commands to operate on model object lists.

Edit Model Object List Panel

The Edit Model Object List panel is an interactive utility for working with lists of model objects. This panel serves as an alternate entry point into your model where you can perform most functions available from the Edit Database Relations panel other than editing relations and creating model objects. You can temporarily transfer to the Edit Database Relations panel from the Edit Model Object List panel by entering YEDTMDL or Y2 on the command line. When you finish your editing, press F3 to return to the Edit Model Object List panel.

The Edit Model Object List panel has a PDM-like interface and has the following main features.

- Multiple views of current model object list
  - Object identification - object name, owner, type, and attribute
  - Audit information - change date, time, user, and type, and impact processed indicator
  - Implementation details - implementation name and date and time of last generation
  - Impact analysis information - date, time, and action required
- Access to model profile
- Options to work with model objects
- Capability of switching between model object lists
- View of detailed description of any model object
- Options and function keys for impact analysis (usages and references)
- Use of user-defined options

For more information:

- On the Edit Model Object Lists panel, see the chapter “Managing Model Objects” in *Generating and Implementing Applications*.
- On model object lists, see the chapter “Managing Model Objects” in *Generating and Implementing Applications*.
- On application areas, see the chapter “Using Your Model” in the *Administration Guide*.

Defining Your Data Model

The purpose of this chapter is to define and create files, fields, and relations based on the entities, attributes, and relationships from your conceptual model. The files, fields, conditions, and relations are the CA 2E basic model design objects that must be defined and created before you can build access paths and functions to operate on your model.

See the chapter “Understanding Your Data Model” for more information on files, fields, conditions, and relations.

This task consists of three steps:

1. Defining Files

2. Defining Fields

3. Entering Relations

You can choose to do one step at a time or to combine all three steps by entering CA 2E relation statements first.

Step 1: Defining Files

You define a file to CA 2E by describing its name and type and its relationship with other files and fields.

Object/Referenced Object File

A file represents an entity within your model; for example, Order. It is referred to as an object in a CA 2E data model.

All of your entity objects must be defined to CA 2E by a file name and file type. For each of the objects you define, CA 2E creates a file. A file can be linked either to another file or to a field through a CA 2E relation. The file or field to which it connects is called a referenced object. Referenced objects must also be defined to as either CA 2E a file or field.

A CA 2E file is defined by several different CA 2E relations. Each database (REF and CPT) file must have at least one key relation. The relations are automatically resolved by CA 2E to determine which fields are to be placed on a file.

File Name

You define a file to CA 2E by describing its name and type. The file name must be unique within your data model. It can contain up to 25 alphanumeric upper or lowercase characters including embedded blanks.

File Type

The file type must be one of the CA 2E valid file types. Depending on how it is intended to be used, a file can have a type of capture (CPT), reference (REF), or structure (STR).

Capture and reference files are database files; structure files are non-database files. Whether a file is capture or reference depends on the role of the fields that make up that file.

See the chapter "Understanding Your Data Model" for more information on using file types.

Capture Files

Capture files should contain regularly recorded transactional data that your application uses.

You should select a CPT file type for files that have a high volume of transactions and require constant update. An example of a CPT type file is an Order file. An order file has many orders that are processed daily.

CA 2E provides three types of default functions for capture files. When you specify a file as a CPT type file, three internal functions are created to allow you to create, change, or delete the records in the file. They are Create Object (CRTOBJ), Change Object (CHGOBJ), and Delete Object (DLTOBJ).

Reference Files

Reference files are master files containing basic data that your application uses.

You should select a REF file type for files that contain non-volatile information; for example, a Customer file. A customer file contains detailed information about a customer such as name, address, telephone.

In addition to the three default functions created for a capture file, a reference file has two other functions that allow you to maintain a file or select a record from a list. They are Select Record (SELRCD) and Edit File (EDTFIL).

Structure Files

A structure file contains a group of fields. These fields can be incorporated into other files by the use of the Includes relation.

For example, you would give the STR type to the Audit Stamp file.

```
Audit Stamp    Has     Update date

               Has     Update time

               Has     User

              Has     Update program
```

Any file within the system that needs the field definitions of the Audit Stamp file can obtain them simply with an Includes relation.

```
Order         Known by                Order number

              Has              Order date

                Refers to       Customer

                  Includes                Audit Stamp
```

Structure file types can use the Has, Refers to, and Includes relations.

CA 2E lets you virtualize fields to an STR file on a Refers to relation. The virtual field in the STR file will not be available as part of the structure when it is included in another file.

The Refers to relation is not moved to the file that includes the structure. There are no referential integrity checks performed for this Refers to relation. The file entry resolved from the Refers to relation is available in the structure.

Structure files are also used to group fields from various files for passing parameters when building functions. When using structure files, all fields of the structure are verified for required and optional checking on device designs.

For more information:

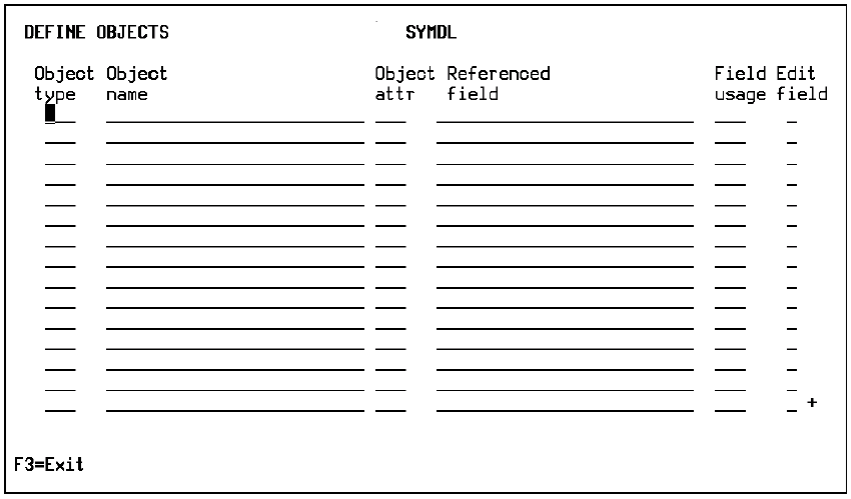
- On parameters, see the chapter "Modifying Function Parameters" in *Building Applications*
- On using arrays as parameters, see the chapter "Defining Arrays" in *Building Access Paths*

Adding Files

Follow these instructions to add files:

1. At the Edit Database Relations panel, press F10.

   The Define Objects panel displays. This panel allows you to create objects. You can define more than one object on the subfile panel for a single Enter key.



Note: If you entered information on the Edit Database Relations panel, some information may appear on the Define Objects panel.

2. Define a file:

   a. Define the object as a file. In the Object type column, enter FIL.

   b. In the Object name column, enter a name for the file.

   c. In the Object attr column, specify the file type by doing one of the following:

- Enter CPT (capture), REF (reference), or STR (structure).
- Select the file type from a list of default attributes. In the Object attr column, enter **?** and press Enter. From the list that appears, select the desired file type.

This procedure automatically creates the Defined as relationship to declare the existence of the file.

Note: The two-character identifying mnemonic lets you define a maximum of 684 files. If you exceed this number CA 2E displays a message instructing you to reset the Last Used File Prefix (YFILPFX) model value and to supply a new object prefix for your model.

To do so run the following commands:

YCHGMDLVAL MDLVAL(YOBJPFX)
VALUE(new-object-prefix)

YCHGMDLVAL MDLVAL(YFILPFX)
VALUE(*RESET)

Note: The first of these commands causes all new objects to begin with the new object prefix. The second command reinitializes the identifying mnemonic for files to AA. As a result, all subsequent file names will be unique

For more information on the YCHGMDLVAL command and model values, see the *CA 2E Command Reference Guide*.

Step 2: Defining Fields

A CA 2E field represents an attribute within a CA 2E data model. A field is the attribute that describes the characteristic of an entity in your conceptual model; for example, Customer Code for Customer, Order Number for Order, or Product Price for Product.

Field Name

A field name must be unique within the data model. It can contain up to 25 alphabetic characters in upper or lowercase, and numeric characters, including embedded blanks.

Field Types

CA 2E provides a number of pre-defined field types that are suitable for different purposes such as values, prices, quantities, and text. You can override the defaults later at the field level and again at the device level. In addition, you can change the supplied defaults or add additional field types of your own.

You define a field to CA 2E by giving it a field name and field type. See the table of CA 2E Field Types that follows for a description of the field types and how you can use them.

Reference Field

The Reference (REF) field type allows you to define one field in terms of another. Reference fields share the same domain, which means that one field takes the same set of values of another field.

Field Types for Referenced Objects

The field type specifies all of the default characteristics for a field.

CA 2E uses field types to make default assumptions about properties of a field. It also uses field types to validate entries.

To describe fields, select one of the field types from the following table.

The following table contains CA 2E field types.

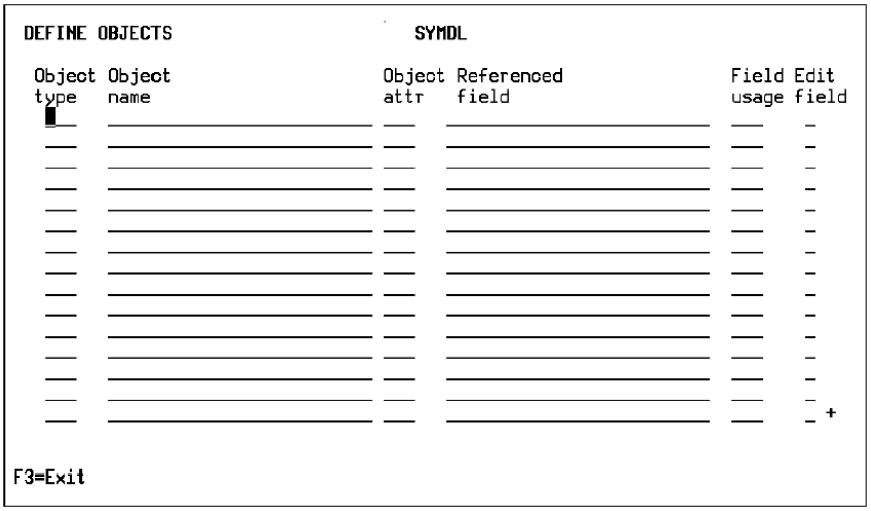
Field Type Name	Description	Type	Length	Example
CDE	Code	A	6	Stock code
DT#	ISO Date	A	10	Order date
DTE	Date	P	7.0	Date of birth
IGC	Ideographic text	A	20	Kanji name
NAR	Narrative text	A	30	Comments
NBR	Number	P	5.0	Number of employees
PCT	Percentage	P	5.2	Profit margin
PRC	Price or tariff	P	7.2	Unit price
QTY	Quantity	P	5.0	Stock quantity
REF	Reference	-	-	Field based on another
SGT	Surrogate	P	7.0	System key
STS	Status	A	1	Discontinued/Current
TM#	ISO Time	A	10	Time process starts
TME	Time	P	6.0	Transaction time
TS#	ISO Timestamp	A	26	Transaction date/time
TXT	Descriptive name	A	25	Product name
VAL	Monetary value	P	11.2	Stock value
VNM	Valid system name	A	10	File name

Specifying Field Types

Follow these instructions to specify field types to newly defined fields.

1. At the Edit Database Relations panel, access the Define Object panel. Press F10.

   The Define Objects panel displays. This panel allows you to create objects. You can define more than one object on the subfile panel for a single Enter key.



   Note: If you entered information on the Edit Database Relations panel, some information may appear on the Define Objects panel.

2. Define a field:

   a. Define the object as a field. In the Object attr column, enter FLD.

   b. In the Object name column, enter a name for the field.

   c. In the Object attr column, specify the field type by doing one of the following:

- In the Object attr column, enter the code for the field type; for example, CDE or TXT.
- Select the field type from a list of default field types. In the Object attr column, enter **?** and press Enter. From the list that appears, select a field type.

   a. In the Field usage column, enter the field usage. The usage defines whether the field can be used as a key (CDE), an attribute (ATR), or a function field.

See the chapter "Understanding Your Data Model" for more information about fields.

Step 3: Entering Relations

CA 2E has two types of relations, file-to-file relations and file-to-field relations, which together consist of eight relation types. To define a relation you must use one of these eight relation types, described later in this topic.

You use CA 2E relations to declare the existence of a file and to describe the connection between files or between a file and a field. The relations you enter for a file are resolved into the fields that are needed to implement that file. Because a file consists of a list of relations, to completely define a file, you must enter all the relations that describe that file.

The Has, Known by, Owned by, and Refers to are CA 2E basic relations that cover most data modeling cases. CA 2E automatically creates Defined as relations for each of the files you define.

CA 2E also resolves a primary key based on the relation types that you enter for a file. It requires a unique key for all database files, which should be the smallest set of fields needed.

Relation Sequencing

CA 2E uses default sequence order for relations. You can override the default sequence by entering different sequence numbers into the Seq column of the Edit Database Relations panel.

The default order and sequence of relations is as follows:

Order/Level	Sequence	Relation
Key	1	Owned by
	2	Known by
	3	Qualified by
Attribute	4	Extended by
	5	Refers to
	6	Has
	7	Includes

The two levels of sequencing are key level and attribute level. You cannot sequence key relations after attribute relations. If you use the same sequence number on different relations within the same level, the order of these relations follows the default ordering.

Sequence numbering follows the collating sequence order of importance. Blank is first, followed by 1, 2, 3, and so on.

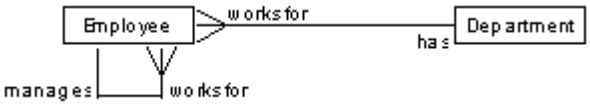
Note: You can add sequence numbers after all relations for the file have been entered. Blank sequence numbers come before numbered sequence numbers.

The Refers to relation for involuted relations should appear after all other relations on the file if virtual fields are to be specified for the relation.

The sequence number you use may have some consequence when you later add virtual fields to a file that references itself.

CA 2E expands the relationship based on which file entries are known at the time the relation is resolved. If a field is to be virtualized but has not been expanded or is not known to the file, the field cannot be virtualized.

For example, Employee Refers to Department and Employee Refers to Employee For Manager. If Department Name is virtualized from Department and Department Name is also desired for Manager's Department, the Employee Refers to Department relationship must be sequenced before the Employee Refers to Employee For Manager relationship.



A file-to-file relationship is expanded to place the fields that are the key of the related file on that file. In the above example, if Department is keyed by Department Code, the relationship Employee Refers to Department is expanded to show the Department Code as a foreign key on the Employee file. Department also has an attribute of Department Name that is to be virtualized onto the Employee file. Upon expansion of the relationships, the virtual fields create file entries. As a result, when the Employee Refers to Department relationship is expanded, the virtual field Department Name also becomes a file entry for Employee.

When the relationship Employee Refers to Employee For Manager is expanded, the key of Employee is another file entry such as Employee Code. For this relationship, you want to virtualize the field Department Name of the manager. Expanding the relationship Employee Refers to Employee For Manager will also include another field: Department Name for Manager.

If the Employee Refers to Employee For Manager relationship is expanded before the Employee Refers to Department relationship, the virtual field Department Name for Manager is not known as a file entry and will not be expanded as a virtual field.

For more information about:

- Procedures, see the chapter on Relation Sequencing (see page 159).
- Virtual fields, see the chapter "Maintaining Your Data Model."

CA 2E Relation Types Charts

Use your conceptual model's ERD as a guide to determine the types of relations needed for your model's files, or consult the CA 2E Relation Types charts that follow. The first chart describes the file-to-file relation types and the second chart describes the file-to-field relation types.

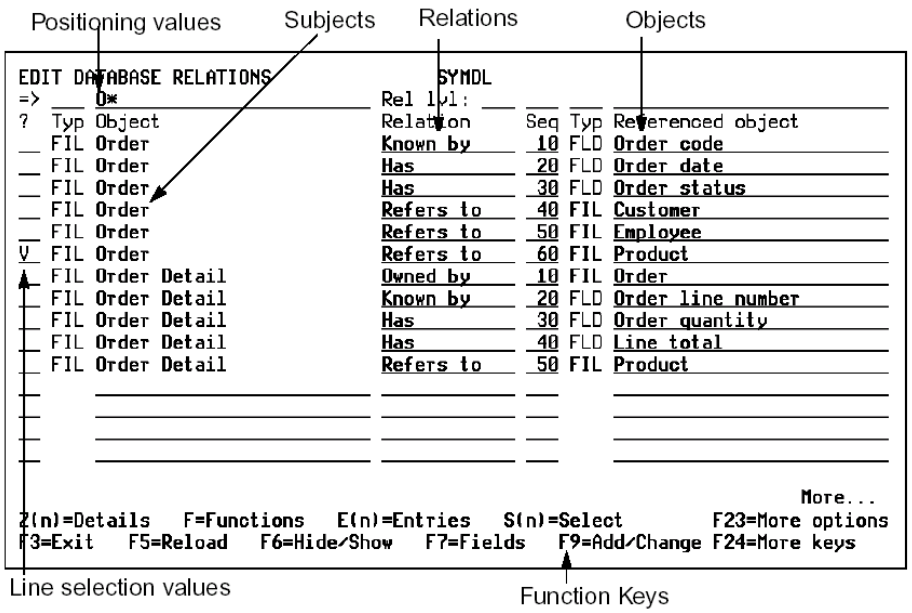
The following table contains file-to-file relations.

Relation	Description
Defined as	Declares that the file exists
Owned by	Specifies that the keys of the owning file are to become major key fields of the owned by file
Extended by	Declares the file to have a one-to-one or one-to-none relationship with another file
Refers to	Causes the key fields of the referenced file to be included as non-key fields on the referring file
Includes	Causes fields from the referenced file or included structures to be included as attributes in the referencing file

The following table contains file-to-field relations.

Relation	Description
Known by	Declares the field to be present as a key field on the file
Qualified by	Declares a field to be present on a file as a key field; is used with continuous variables
Has	Declares the field as a data field on the file

To enter relationships, use the Edit Database Relations panel:



1. Enter the object name. The object name is always a file name.
2. In the relation column, enter the relationship.
3. Enter the referenced object name:
   - For a file-to-file relation, enter a file name.
   - For a file-to-field relation, you must enter a field name.

WARNING! Do not use CA 2E shipped fields, those beginning with '*', for defining file-to-field relations.

Define all objects and referenced objects. If you do not define an object or referenced object, CA 2E highlights the name and sends a message letting you know that it needs to be defined.

To define these objects, access the Define Objects panel by pressing F10. The Define Objects panel displays with some information already entered.

For undefined objects, the object type is FIL and the file name is the object name. In the Object attr column, enter the file type (CPT, REF, STR).

For undefined referenced objects:

- The object type is based on the relation. If the relation is file-to-file, the type is FIL. If the relation is file-to-field, the type is FIL or FLD.
- The object name is the referenced object name.
- If the object type is FLD and the relation is a Known by relation, the field usage is CDE. If the relation is a Qualified by or Has relation, the field usage is ATR.

For the object attribute:

- If the object type is FLD, enter the field type in the Object attr column.
- If the object type is FIL, enter the file type (CPT, REF, STR) in the Object attr column.

You can use this method for defining files, fields, and relations in combination with other described methods. If you enter all information on the Edit Database Relations panel, CA 2E defaults the values to the Define Objects panel. Complete the definition for the objects on the Define Objects panel.

See the chapter "Understanding Your Data Model" for more information on CA 2E relations and examples.

This topic provides detailed information for working with file entries.

File entries are resolved from CA 2E relations that you use to describe file relationships in your model. An entry indicates the presence of a field on a file. A relation may imply that one or more fields are to be created for your file.

The entries, excluding virtual field entries, indicate the fields to be present in a physical file.

Levels of Entry

The entries of a file arise from three different levels at which you specify relations for the file:

- File level—all those fields resulting from the resolution of file relations.
- Access path level—all those fields resulting from the resolution of access path relations. They must be either inclusive of or be a subset of the entries from the file relations level.
- Device file level—those fields resulting from the resolution of device file relations. Entries resolved at this level are a subset of the access path level entries, which may not include every relation of the file.

  You can add additional entries to device designs for function fields.

Entry Types

Entries are classified into three categories depending on the types of relation from which they are resolved:

- Key field entries
- Attribute entries
- Virtual field entries

There are no entries resolved from definition, or Defined as, relations. A Defined as relation is used simply to define a file.

Key Field Entries

These entries arise from the resolution of CA 2E key relations, such as Owned by, Known by, and Qualified by.

For example, if you specify:

Customer is Known by Customer code

the Customer code field will be present as a key field on the Customer file.

Attribute Field Entries

These entries arise from the resolution of CA 2E attribute relations, which are Refers to, Has, and Includes**.**

For example, if you specify:

Customer **Has** Customer name

the Customer name field will be present as an attribute field on the Customer file.

Virtual Field Entries

These entries arise from the specification of virtual fields on file-to-file relationships expressed by the Refers to, Owned by, or Extended by relations.

For example, if you have the relation:

Order Refers to Product

where Product name is specified as a virtual, Product name will appear on the Order file as a virtual field.

See the chapters "Understanding Your Data Model," and "Maintaining Your Data Model" for more information on how to specify virtual fields at file relations level.

Overriding Entries

You can override CA 2E default entries of a file with replacing, sharing, and redirection.

You can override default entries for those entries arising from certain relations: Owned by, Refers to, Extended by, and Qualified by relations.

Replacing Entries

When more than one instance of a field is defined for a file, separate fields and names are created by default. CA 2E automatically defines and creates the necessary additional field based on the existing field.

You can specify an alternative field to replace the one CA 2E supplies. The new field must have the same domain as the one it replaces. It must be defined as a field of REF type, with a definition based on the replaced field.

If you already have a field defined that you would rather use, you can specify it on the Display Referenced Field Details panel (Replace field). This panel shows all the eligible fields for an entry. From here, you can transfer to the Define Objects panel to define new fields, based on the field to replace.

Sharing Entries

Key fields (identifiers) can be shared between file entries arising from the resolution of the Owned by and Refers to relations.

All files that you use as targets of sharing must always be defined as relations above the relation you are defining. Never share relations that are lower, and do not move shared relations lower.

Sharing is performed by matching the keys of the file named in the sharing parameter to the fields that already exist in the file. Thus you can control the fields shared by proper sequencing of the relations. When you name a file in the sharing parameter that is defined as a prior relation in the file, sharing uses the keys defined for that relation in preference.

For example:

Order Detail Refers to Item Master

For: Ordered

where Item No. (ordered) is the key

Order Detail Refers to Item Balance

For: Shipped, Sharing: *NONE

where Item No. (shipped) and Warehouse No.

are the keys

Order Detail Refers to Shipping Instructions

Sharing: Item Balance

where Item No. (shipped) and State Code

are the keys

The explicit reference of Sharing Item Balance ensures that the Item No. of the item being shipped is used as the key to the Shipping Instructions, rather than that of an item ordered even though the Ordered Item No occurs higher in the entries.

See the chapter "Understanding Your Data Model" for more information on sharing entries between relations.

Redirection

Normal resolution of a CA 2E relation causes one or more fields to be added as entries to a file. You can override this resolution by redirecting a relation entry. Redirecting means that you specify that the source of a field value, needed to implement an instance of the relation, is to be supplied from another field of the same type already present on the file.

A source field may itself be either a virtual or non-virtual field. Advantages are as follows:

- There is less need to carry redundant fields on a record purely for the purpose of supplying keys to access another file.
- You can specify redirection of key fields to other instances of the same (base) field on a record.

Redirecting Entries

You can redirect entries resolved from a Qualified by relation or a Refers to relation.

Redirection of key fields resolved from a Qualified by relation provides a central mechanism for indicating to functions that when records are to be retrieved from a file to satisfy a relation, the retrieval is to be done on the basis of a nearest match rather than an exact match. For instance, you would use the prices in effect on a given date to price an order on that date for each product.

Retrieval may be done on a basis of nearest less than or nearest greater than depending on the value specified for the Sharing field.

The entry to be redirected must be sequenced after the relation entry that provides the source field for the redirection. You can specify sequence numbers on the Edit Database Relations panel to override the default order of relations.

Never redirect relations that are lower or move these relations so they are lower. If relations are not sequenced properly and you select a field for redirection, the selection will not take effect.

For more information on relation sequencing, see the chapter Creating/Defining Your Data Model.

The following discussion covers the two types of redirection: redirection of qualifier fields and redirection of key fields.

Redirection of Qualifier Fields

A qualifier field is resolved from a Qualified by relation. Redirection of qualifier fields allows you to specify that a relation between two files is satisfied not by an exact match of values, but by the nearest match. The redirected field gives the search value.

This redirection is particularly useful when you deal with variables such as prices and discounts.

You can redirect entries arising from qualified fields to any other fields of the same attribute type that are present in the referencing file. For example, dates can be redirected to other dates, numbers to other numbers, and values to other values.

If the length of the overriding field is not the same as the redirected field, a truncated value is used.

For qualified redirection, a field that has been redirected is *not* dropped from the model file entry list or from the generated physical file.

You can use qualified redirection only on fields defined as qualified keys on the referenced file by means of a Qualified by relation.

Example of Redirecting Qualifier Fields

A Product price could be defined as follows:

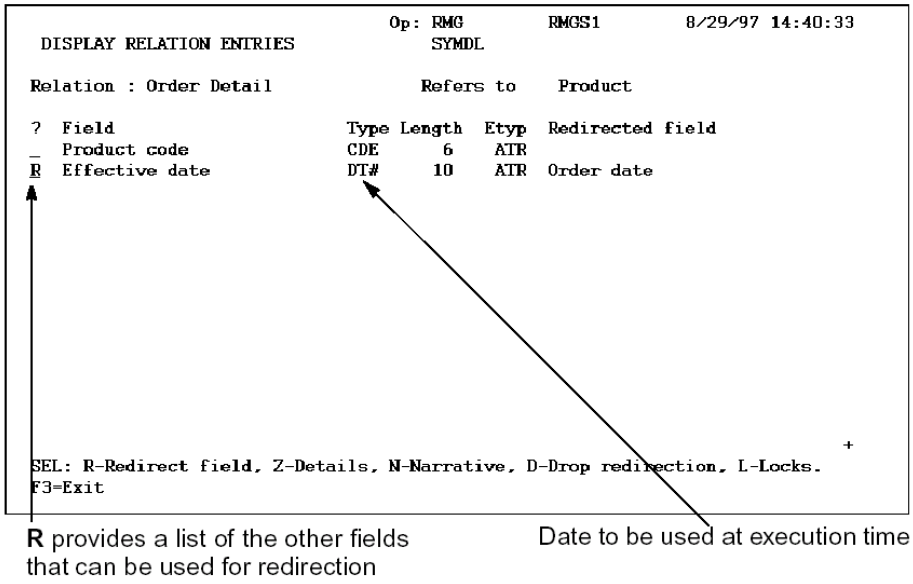
```
FIL Product price     REF    Owned by             FIL    Product              REF

FIL Product price    REF     Qualified by    FLD     Effective date  DT#

FIL Product price    REF     Has                     FLD     Product price   PRC
```

When referring to the Product price in relations that describe other files, you may redirect the source of the Effective date as follows:

```
FIL  Order detail   CPT     Refers to       4 FIL   Product price     REF

                Order date          RDR  Effective date    DT#
```

Example of Redirecting a Reference to a Qualified File

When pricing orders, you always want the current price to be used. If product prices do not change every day, each product price record represents not an individual price on a particular day, but a price that is current over a period. When retrieving a price record you do not necessarily retrieve a record exactly matching the date; instead, you want the nearest record. To achieve this, redirect the qualifier reference, using the Display Relation Entries panel. To access this panel, place E0 against the Refers to relation on the Edit Database Relations panel. The following is a sample Display Relations Entries panel.



Fields may only be redirected to other fields on the access path that are of the same field type; for instance, both DT# fields.

If Order date is redirected to Effective date, at execution time:

- The Order date is used to look for the price currently in effect. The price record found is that with the nearest date previous to the Order date. Thus if an Order date of 11/29/93 is used to look for the price for Product 00002 in the following table of prices, the price record in effect from the 11/15/93 would be found:

Search Values

Product	Order Date
0002	**93/11/29**

File Values

Product	Effective Date	Price	
0001	1992-09-05	20.00	
0001	1992-10-01	21.20	
0002	1993-10-01	20.00	
0002	**1992-11-15**	**23.00**	**FOUND**
0002	1993-12-10	22.50	
0003	1992-09-05	44.25	

- The date of the price is placed on the Order record.

  Note that if new price records are added, such as 11/18/93, 22.75, pricing of the existing order is not affected unless it is specifically repriced. However, new orders are priced at the new rate automatically.

  Note also that, if product prices change every day, you would not need to redirect the reference since there would be a record present for each product for each date. Each product price would then represent a discrete value rather than a continuum of values.

Procedures for Working with Entries

This section includes step-by-step instructions for

- Displaying file entries
- Replacing file entries
- Displaying and redirecting relation entries
- Modifying For Text and Sharing entries

Display File Entries

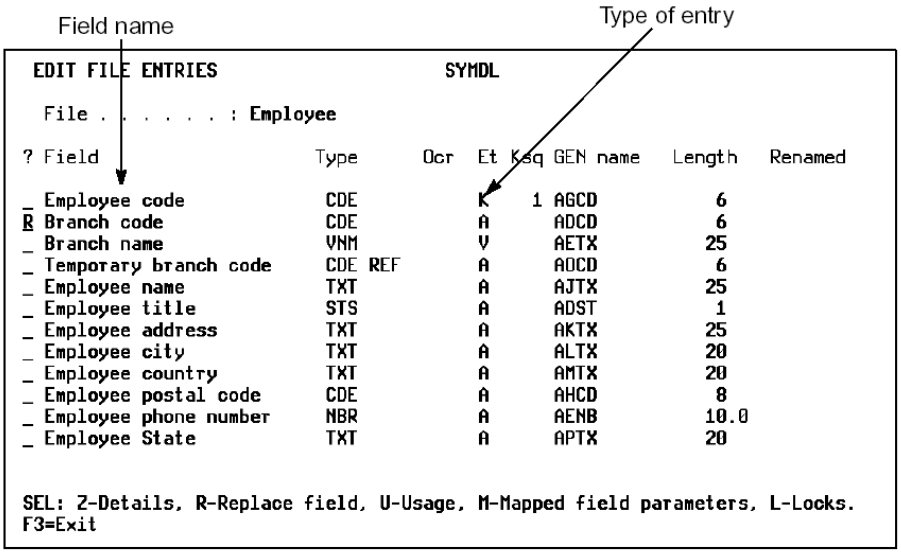
Displaying the entries for a CA 2E file gives you a means of examining the fields that will be present in the physical file. The physical file is used to implement your file CA 2E. To display file entries:

1. On the Edit Database Relations panel, enter E next to the file for which you want to display entries (for example, Customer) and press Enter.

   The Edit File Entries panel displays the entries for the file you selected. A sample panel appears below.

2. To exit, press F3 and return to the Edit Database Relations panel.

The Edit File Entries Panel



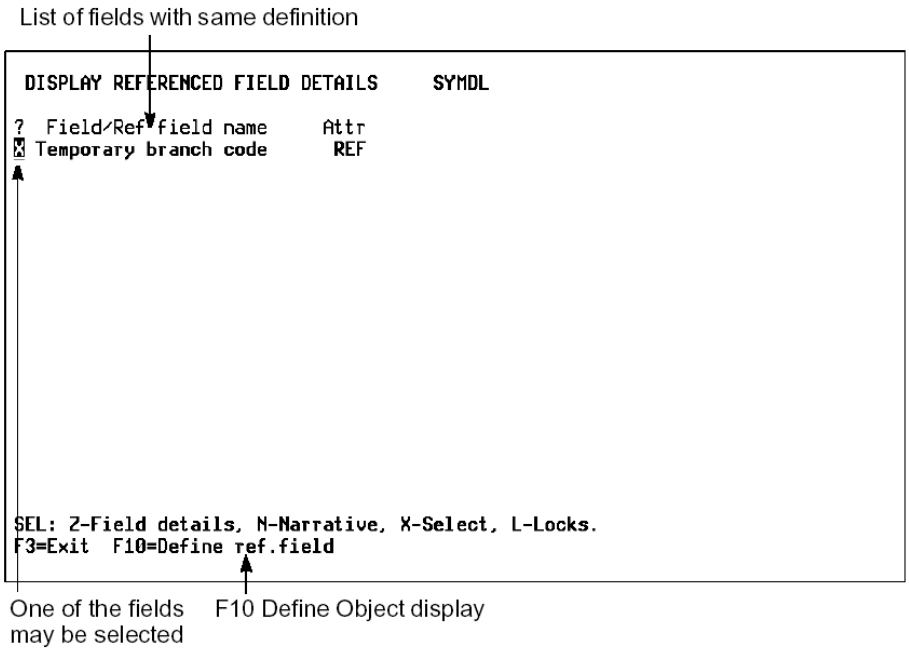
Replace File Entries

1. On the Edit File Entries panel, enter R next to the field you want to replace.

   The Display Referenced Field Details panel appears. This display lists all the fields with the same definition that are eligible fields for an entry. You may select one of them. A sample panel appears below.

2. Select the field you want by typing X next to it. This field replaces the field you indicated in step 1.

Display Referenced Field Details Panel



From this panel you can transfer to the Define Objects panel to define new fields. The Define Objects panel will have transferred to it all the required information. You have only to enter the field name.

Display/Redirect Relation Entries

You can display and redirect entries resolved from Refers to relations only. Follow these steps to display and specify redirection of relation entries.

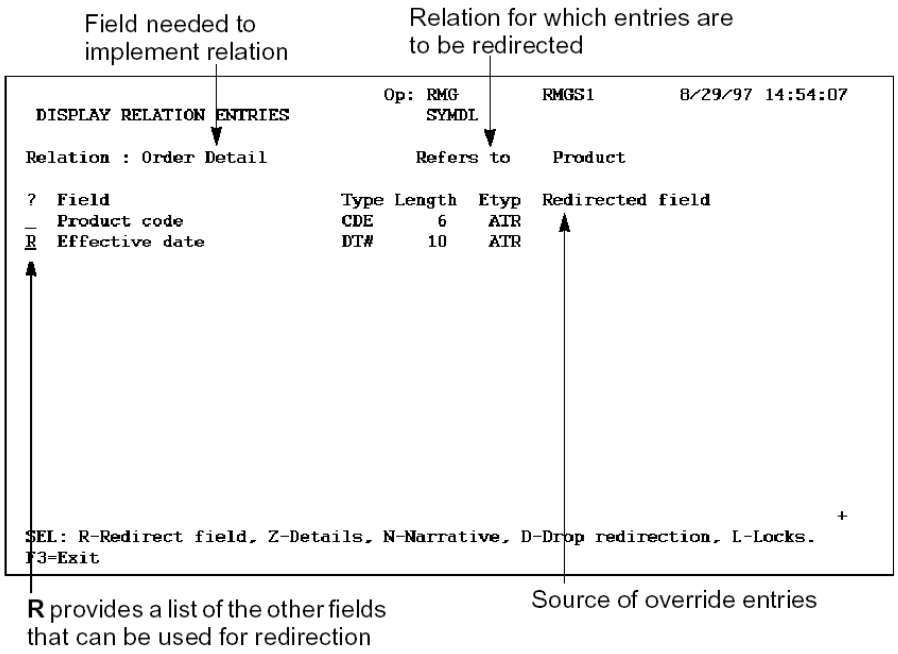
1. From the Edit Database Relations panel, enter E0 next to the Refers to relation and press Enter.

   CA 2E displays the Display Relation Entries panel. A sample panel appears below.

2. Enter R next to the desired field to access the Edit Redirected Fields panel.

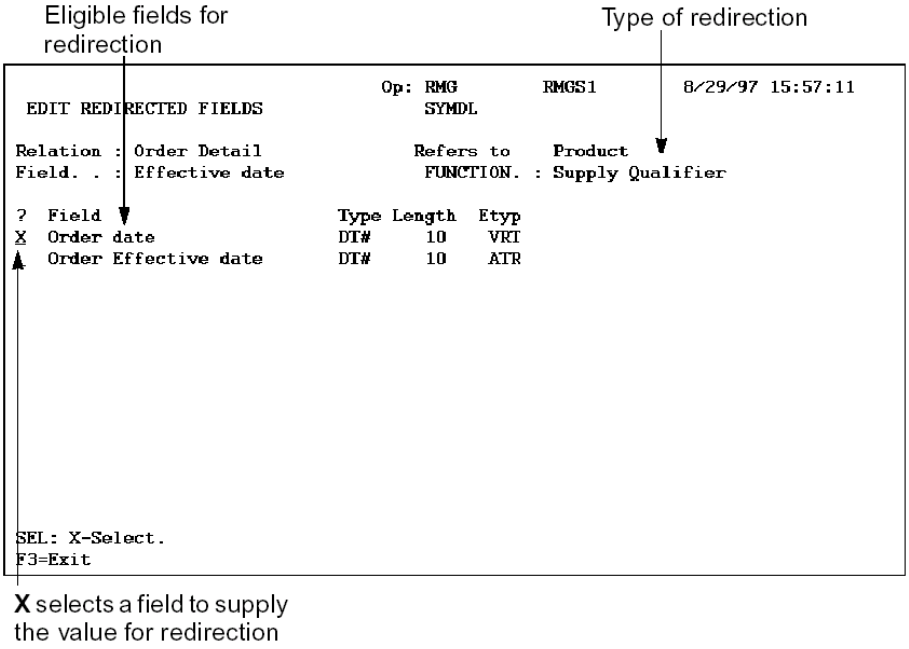
   The Edit Redirected Fields panel shows, for a given relation entry, all the possible other entries to which the relation entry can be redirected. You can select any one of the indicated fields to specify redirection to that entry. A sample appears on the next page.

Display Relation Entries Panel



1. Enter X next to the field that you want to use to supply the value for redirection and press Enter.

Edit Redirected Fields Panel



Modifying For Text and Sharing Entries

Follow these steps to modify For Text and Sharing entries.

1. Position the Edit Database Relations panel by entering the file name in the Object positioner field at the top of the panel and press Enter.
2. Place a plus sign (+) in the subfile select field next to the relation and press F5. You are allowed to expand Owned by, Refers to and Qualified by relations.
3. The panel will be expanded to allow you to modify the For Text and Sharing entries.

Chapter 5: Maintaining Your Data Model

This chapter provides detailed instructions for you on the various tasks needed to maintain a data model in CA 2E.

You perform the tasks described in this chapter to add more information to a newly created model or to modify an existing one. If you are in the process of building a model, complete the tasks in the order they are listed.

This section contains the following topics:

Displaying File Entries

In the previous chapter "Creating/Defining Your Data Model," you entered the definitions of your files, fields, and their relations.

Based on the relations you entered, CA 2E creates entries for your files. File entries are fields that are resolved from CA 2E relations for a file and used to implement that file. A relation may imply more than one entry on a file.

You may want to view those entries now so that you can modify the fields or add new information to suit your model's needs.

See the chapter "Creating/Defining Your Data Model" for more information about entries.

Edit File Entries Panel

You can view the entries of a selected file using the Edit File Entries panel. This panel shows the field names, field type, the CA 2E implementation name for the field, the default CA 2E field length, and whether the field is a key field or an attribute. It also shows the key sequence (Ksq). This sequence dictates the order in which the fields compose the primary key. You cannot change any of this information while you are at this panel.

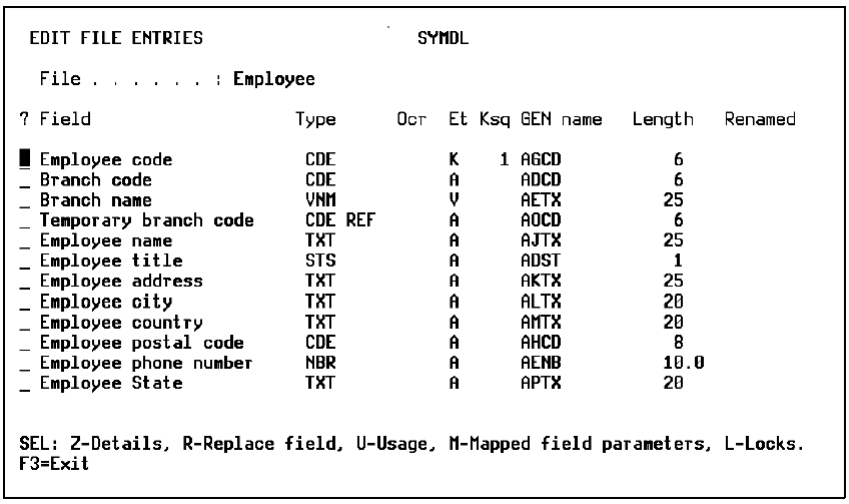
For more information on adding or changing fields, see the Adding/Modifying Field Information section in this chapter.

Display File Entries

To display file entries follow these steps.

1. On the Edit Database Relations panel, type E next to the file for which you want to display entries and press Enter.

   The Edit File Entries panel displays.



```
EDIT FILE ENTRIES                    SYMDL

  File . . . . . : Employee

? Field                    Type      Ocr  Et Ksq GEN name   Length   Renamed

_ Employee code            CDE            K    1 AGCD          6
_ Branch code              CDE            A      ADCD          6
_ Branch name              VNM            V      AETX         25
_ Temporary branch code    CDE REF        A      AOCD          6
_ Employee name            TXT            A      AJTX         25
_ Employee title           STS            A      ADST          1
_ Employee address         TXT            A      AKTX         25
_ Employee city            TXT            A      ALTX         20
_ Employee country         TXT            A      AMTX         20
_ Employee postal code     CDE            A      AHCD          8
_ Employee phone number    NBR            A      AENB       10.0
_ Employee State           TXT            A      APTX         20

SEL: Z-Details, R-Replace field, U-Usage, M-Mapped field parameters, L-Locks.
F3=Exit
```

   Note: CA 2E has created these entries for your Employee file as shown on the panel and indicated which entry is a key field (K), an attribute field (A), or a virtual field (V).

   A key field entry is resolved from the Owned by, Known by, and **Qualified by** relations. An attribute field entry is resolved from a Has, Refers to , or Includes relation. The key order is indicated under the Ksq column.

   CA 2E also gives each entry an implementation name and a default length according to the field type you entered for these fields.

2. Press F3 to return to the Edit Database Relations panel.

You can document and obtain hard copy printouts of the files, fields, and relations that you entered for your model by using CA 2E documentation commands.

See the chapter "Documenting Your Data Model" for more information on documenting your data model.

Adding/Modifying Field Information

This section lists the tasks to add new information or to modify existing information for the fields that CA 2E creates for your model's files.

Using the Edit Field Details Panel

You can access the Edit Field Details panel in the following ways:

- From the Edit Database Relations panel, type Z2 on the relation with the field as the referenced object and press Enter.

  The Edit Field Details panel displays.

- From the Edit Database Relations panel, do the following:

  a. Press F7 to display all of the fields on the Edit Fields panel.

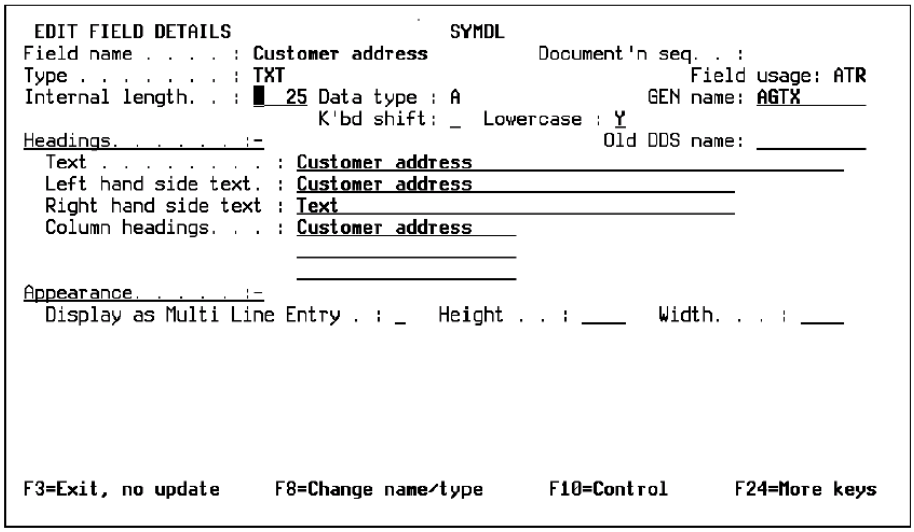
  b. From this panel, select the desired field and access the details by typing Z next to the field, and pressing Enter.

- From the Edit Model Object List panel, enter option 2 for the selected field.

The Edit Field Details panel displays.

The Edit Field Details panel allows you to modify field information. The values for the field were originated from default values of the field type. These values can be modified to change the characteristics of the field. The changes you make to a particular field will be available throughout the model.

Press F10 to view the appearance fields instead of the field control information.



The following table lists the overrides for CA 2E default field values for field types.

Data Attribute	Default	Model Level	Field Level	Device Level
Field length	By type	Y	Y	N
System data type	By type	N	-	-
Keyboard shift	By type	Y	N	N
Implementation name	By type	-	-	N
Text headings	By type	-	-	Y
Left hand side text	Field name	-	-	Y
Right hand side text	Field type`	Y	Y	Y
Column headings	Field name	-		Y
Allow lowercase	By type	Y	Y	N
Mandatory fill	N	Y	Y	N
Valid system name	VNM only	Y	Y	N
Mod 10/11 check	N	Y	Y	N
Field exit option	By type	Y	N	Y
Check condition	By type	-	Y	Y
Translate values	STS only	N	Y	N
Edit codes: Input	By type	Y	Y	Y
Output	By type	Y	Y	Y
Report	By type	Y	Y	Y

Change Field Name and/or Type

Keep these points in mind when you rename fields:

- The text and column headings automatically change.
- You cannot rename two fields with the same name.

To change the type and/or name of a field:

1. From the Edit Database Relations panel, press F7 to display the Edit Fields panel.
2. Zoom into the details of the field by entering Z next to the field and pressing Enter. The Edit Field Details panel displays.
3. Press F8 to change the name or type. The cursor will be on the name entry area.
   a. To change the field name, type a new name over the current one and press Enter.
   b. To change the field type, key a new field type over the current one and press Enter. You may also place a ? to select from the list of field types.
4. Press F3 to return to the Display Fields screen. The new field name and/or type you entered is shown on the panel.

   Press F3 to return to the Edit Database Relations panel.

Change Field Length

You can change the length of a field from the Edit Field Details panel. Type the new length over the existing one and press Enter.

Note: You cannot change field length for DTE fields.

Add Narrative Text

To add narrative text to a field:

1. Access the Edit Narrative Text panel in any of the following ways:
   - On the Edit Field Details panel, press F20.
   - Type N2 next to a relation, with the field from the Edit Database Relations panel as the referenced object.
   - Use selection options 21 or 22 from the Edit Model Object List panel.
2. Add any text, notes, or descriptive information you want to include for the field at this panel.

This becomes the generated help text or design documentation.

For more information:

- About narrative text, see the chapter "Using Your Model" in the *Administration Guide*
- On including narrative text in documentation listings, see the chapter "Documenting Your Data Model" in this guide

Change Field Text and Headings

You can modify the text and headings of a field from the Edit Field Details panel. The headings are used on reports and panels as prompts for the field. They have initial values of the field name.

To change field text and headings from the Edit Field Details panel, type the new text over the existing text and press Enter.

Change Valid System Name (VNM)

The Valid System Name value for fields of VNM type is modifiable. To change field Valid System Name (VNM), on the Edit Field Details panel, enter one of the allowable values (Y or Blank).

See the chapter "Understanding Your Data Model" for more information on field types and how to modify field type values.

Adding/Modifying Conditions

A condition specifies the value or list of values a CA 2E field may take. You can add, change, or delete conditions using the Edit Field Conditions and the Edit Field Condition Details panels.

Condition Types

A condition type specifies the type of validation rule it imposes.

Field conditions can be used to

- Validate the entry of data
- Select or omit data in access paths
- Specify processing conditions in a function that operates on the data

For more information about using conditions with access paths and functions, see the *Building Applications* chapter "Modifying Action Diagrams"

Select one of the following four types of conditions:

- VAL to specify single values
- LST to specify a list of values
- CMP to specify an arithmetic comparison for a field
- RNG to specify the range of valid values for a field

VAL and LST are used for status fields; CMP and RNG are used with non-status fields.

For this task, you will use conditions to

- Validate the entry of data
- Specify default values when adding records to a file

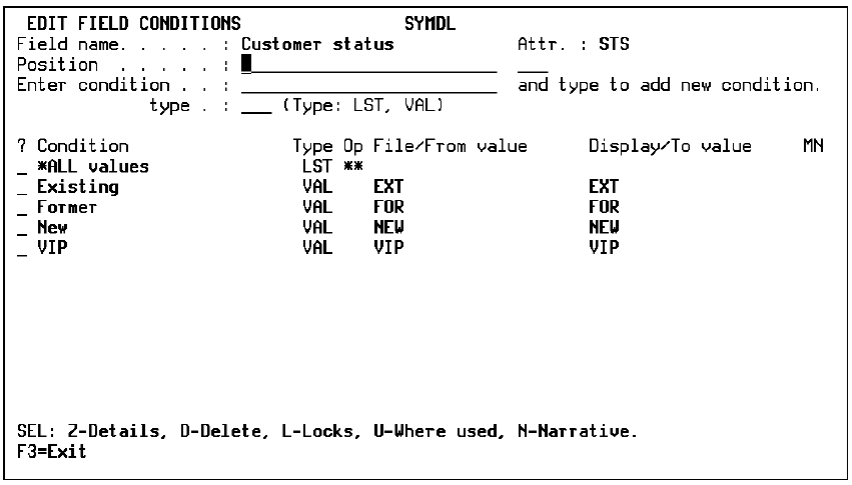
Using the Edit Field Conditions Panel

The Edit Field Conditions panel shows all the conditions that exist for a selected field. Use this panel to add or modify conditions.

1. From the Edit Database Relations panel, do either of the following:
   - Type Z2 next to the relation with the field you want to add conditions.
   - Press F7 to get a list of fields, then type Z next to the field to which you want to add conditions, then press Enter.

     The Edit Field Details panel displays.
2. Press F9 to obtain the Edit Field Conditions panel.

   The Edit Field Conditions panel displays the conditions for the selected field.



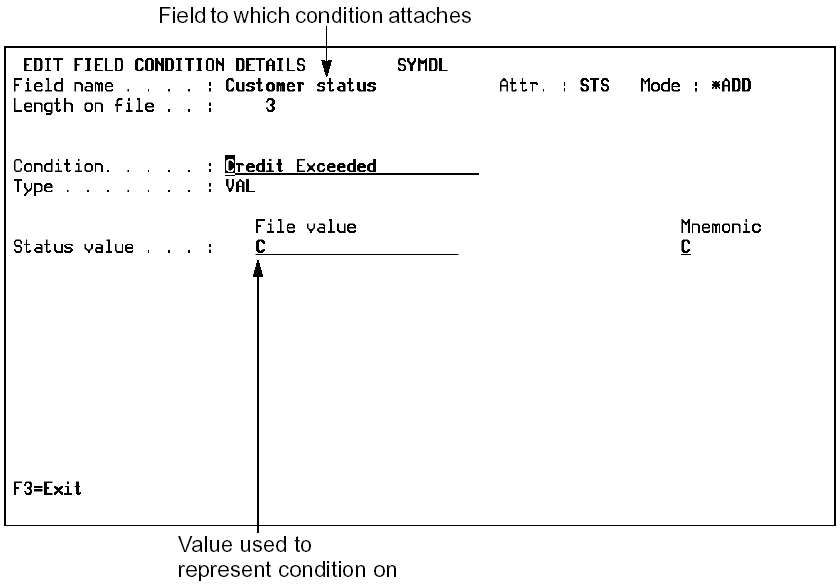
Add New Conditions

When adding a condition, you must add the condition type, the meaning of the condition, and the value used to describe the condition in the file.

1. From the Edit Field Conditions panel:

   Type a name and a condition type (VAL, LST, CMP, or RNG) and press Enter.

   The Edit Field Conditions Details panel displays.



For STS fields:

- If type is VAL, and Translate condition value is Yes, enter the display and file (storage) values for the condition.
- If type is VAL, and Translate condition value is No, enter the file (storage) and mnemonic values for the condition.
- If type is LST, select the existing conditions for the list by typing **+** next to each selection. Deselect conditions that are currently attached to the list but no longer required by typing **-** next to each selection.
- If type is LST, specify one of the following prompt functions. These functions provide a method to display and select the allowed values list when F4 or **?** is used for the field.

- Condition Values Displayer
- Drop Down List

For non-STS fields:

- If type is CMP, enter the compare operator and the compare value.
- If type is RNG, enter the From compare value and the To compare value.

Note: You can add additional VAL, CMP, and RNG conditions while on the Edit Field Details panel.

Press F3 to return to the Edit Field Conditions panel.

The conditions you have added are shown on the panel.

To Modify Existing Conditions:

1. From the Edit Field Conditions panel, select a condition by typing Z next to the condition you want to modify and press Enter. The Edit Field Conditions Details panel displays.
2. Change the name of the condition using F8 to make the field name modifiable**.**
3. Change the existing condition information and press Enter.

   The Edit Field Conditions panel displays.

For VAL conditions, the value listed under File/From Value is stored in the file; the value under Display/To Value is what appears on all panels and reports. For example:

- Text Description: Open Order
- File/From Value: O - stored in file
- Display/To Value: OPN - shown on panel
- Mnemonic - shown on panel

File/From Value and Display/To Value are often the same.

Note: CA 2E automatically adds a special condition for the *ALL values LST condition of a STS field.

Using VAL and LST Conditions

VAL conditions on status fields describe a single condition for that field, such as Open or Invoiced. It is often desirable to describe a set of these conditions and address them as a single condition. For example, Open, Picked and Shipped conditions might constitute the condition Active. To do this, CA 2E provides the ability for a set of VAL conditions to be grouped under a single LST condition.

By default, all VAL conditions are included in the special LST condition *ALL values. You may create other LST conditions from any combination of VAL conditions.

LST conditions also permit the same VAL to be relabeled as a different condition. In this case, the LST has only one entry, that of the VAL. This is particularly useful with subfile selectors and function keys where a single VAL (such as "A" or "F8") may define different conditions on different panels.

To accomplish this you may consider changing all your VALs on the Subfile selector field SFLSEL to condition such as:

VAL	Condition	Value
VAL	A	A
	B	B

and to then use a single entry list to define the actual condition. For example:

LST	Condition	Entry
LST	Add	A
LST	Activate	A
	Allocate	A

Validating Field Entries Using Check Condition

To validate data entry for a field, CA 2E uses a check condition which initially defaults to *NONE. The default indicates that any value is allowed for the field, even if conditions have been defined for it.

You can override this default with one of the conditions you have added to a field to impose specific validation. The field value will be validated against the condition for the field. If the value does not meet the criteria of the condition, an error message will be displayed.

Note: Although you have defined conditions, if you leave check condition to *NONE, checking will not occur at this level. It can still take place at the device level.

For STS fields, the check condition must be a LST condition. As an example, follow these instructions to specify the *ALL values condition to validate a status field.

- On the Edit Field Details panel, type *****ALL values in the Check condition field and press Enter.
- You can also select a condition from a list of existing conditions by typing **?** in the Check Condition field on this panel.

Changing Default Conditions

You use a default condition to provide a value to a field when there is no value entered for the field. The default condition specifies the value to store for the field.

The allowable types for the Default condition are

- VAL for STS fields
- CMP with an *EQ operation for all other fields

The Default condition has a value of *NONE, defaulted by CA 2E. You can change the condition to any valid condition available for the field.

1. On the Edit Field Details panel, you can either:
   - Type the name of the condition to be used for the Default condition.
   - Or select a condition from a list of conditions for that field with **?** and press Enter.
2. Press Enter.

Changing Translate Condition Values

The Translate Condition values field is available only for STS fields. This is used to specify that the field has a display value that can be different than the stored value. To give this capability to a STS field:

- You must provide a Check condition of LST for the field. Even if the Check condition is not correct at this time, you must provide a dummy.
- You can then change the Translate Condition value from blank to Y.

To access the Edit Field Conditions panel and change Translate Condition values field:

1. From the Edit Database Relations panel:
   a. Press F7 to access the Display Fields panel.
   b. Type Z against the desired field to access the Edit Field Details panel.
2. From the Edit Field Details panel, type ? for the Check Conditions field and Y for the Translate Cnd Values field.

   Placing a **?** displays the Edit Field Conditions panel. If there have not been any conditions defined for the field, the panel will not show any conditions. In this case, you need to define a condition of type VAL for the field.

   This condition will be a single value condition since the translation of values has not yet been defined for the field. After you create the condition, a new condition should appear, the *ALL values condition. It was created automatically by CA 2E.
3. Select the desired condition by typing X next to a LST Condition field on the Edit Field Conditions panel and press Enter.
4. Press F9 from the Edit Field Details panel to display the Edit Field Conditions panel.
5. Create the desired condition by typing the name and type of the condition and press Enter.

   The condition of type VAL must have an internal and an external value.
6. On the Edit Field Conditions Details panel, enter the values and press Enter.

Note: If an initial condition was created to form the *ALL values condition, this initial condition has the same internal and external value. You can change or delete this condition if desired.

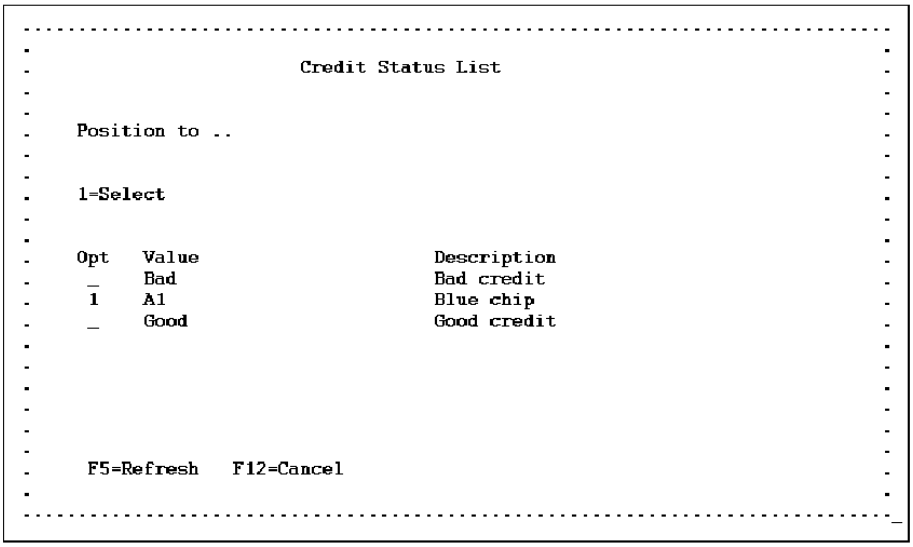
Converting Conditions to List of Values

CA 2E generates source code to call a select facility for any status fields that appear as input-capable fields on a function device design. If you prompt a status field that requires data input, the select facility displays the allowed external values.

CA 2E provides the Display Values List panel.

For example, to display a list of values for the Credit Status field, press F4 while positioning your cursor on the Credit Status field on the device file of a CA 2E generated program.

You will see the following panel:



Place 1 to select a value.

You can create the values list file from a CA 2E model using the Convert Condition Values (YCVTCNDVAL) command.

For more information on using the YCVTCNDVAL command see the *CA 2E Command Reference Guide*.

Adding/Modifying Virtual Fields

You add virtual fields to a file-to-file relation, such as Refers to, to indicate which item of data can be obtained through the relation. Virtual fields provide a way to view information from another related file without having the information physically exist in the related file.

Virtual Fields and Access Paths

Virtual fields are defined through relations to a file. These virtual fields are available to access paths built over the file. However, additional virtual fields cannot be added at the access path level.

For more information on access path virtual fields, see the *Building Access Paths* chapters "Modifying Access Paths" and "Tailoring For Performance."

Example of Using Virtual Fields

Because the Order detail file Refers to the Product file, you can specify any Product detail, such as Product name or Pack size, as a virtual field on the Order detail file. For example:

- If you define a Product as follows:

FIL	Product	REF	Known by	FLD	Product code	CDE
FIL	Product	REF	Has	FLD	Product name	TXT
FIL	Product	REF	Has	FLD	Pack size	QTY

Any file that has a Refers to relationship with a relation that refers to Product can include any non-key field of Product as a virtual field. For example, an order detail line can include Product name and Pack size as virtuals:

FIL	Order detail	CPT	Refers to	FIL	Product	REF
				VRT	Product name	TXT
				VRT	Pack size	QTY

By indicating a field on the Product file (referenced file) as a virtual field, you allow the system to make the data contained in this particular field of the Product file available for functions that operate on the Order Detail file (referencing file).

In this case, the Order Detail file may include any of the non-key fields of the Product file as virtual fields.

When you specify virtual fields for a relation, CA 2E generates the necessary source to join the files that actually contain the virtual field to the related file. For DDS, this process is usually implemented through the use of an i OS join logical file. For SQL, a view over multiple tables is used. Because of i OS limitations, CA 2E generates special logic to support virtual fields in SPN access paths.

You can only add virtual fields to relations that connect a pair of files through the relation types Owned by, Refers to, and Extended by.

When you specify virtual fields for a file that references itself, sequence the Refers to as the last relation.

Virtualizing Virtual Fields

The fields you have chosen as virtual fields may themselves be virtual fields on the referenced file.

For more information see the chapter "Creating/Defining Your Data Model."

For instance, if in the above example Product Refers to Quality, with Quality name specified as a virtual field, then Quality name may be a virtual field for Order detail as well.

FIL	Quality	REF	Known by	FLD	Quality code	CDE
FIL	Quality	REF	Has	FLD	Quality name	TXT

FIL	Product	REF	Refers to	FIL	Quality	CDE
				VRT	Quality name	TXT

FIL	Order detail	REF	Refers to	FIL	Product	CDE
				VRT	Product name	TXT
				VRT	Pack size	QTY
				VRT	Quality code	CDE
				VRT	Quality name	TXT

You can specify as many levels of virtual fields as allowed by i OS for levels of database join. Depths of three or more are not recommended.

Follow these instructions to add virtual fields to a file relation or modify existing virtual fields.

1. On the Edit Database Relations panel, type V next to the file relation for which you want to add virtual fields and press Enter.

   The Edit Virtual Field Entries panel displays. This panel shows a list of fields of the referenced file.

2. Type + next to the field that you want to be a virtual field and press Enter .

   You can add one or many virtual fields for a relation.

   Pressing Enter confirms your selection. Note that an ***** (asterisk) has been placed in the selection column, indicating the field is now a virtual field.

   To remove a virtual field, type a – next to the selected field.

Related Procedures for Maintaining Your Model

Use the following procedures to perform maintenance tasks involving files, fields, conditions, and relations.

Note that the maintenance described in this subtopic can also be accomplished using options on the Edit Model Object List panel.

For more information on the Edit Model Object List panel, see the *Generating and Implementing Applications* chapter "Managing Model Objects."

Files

Add Narrative Text

Follow these steps:

1. On the Edit Database Relations panel, type N next to the file. The Edit Narrative Text panel displays.
2. Fill this panel with the text, notes or any descriptive information you want to include.

For more information:

- On narrative text, see the *Administration Guide*
- On how to include narrative text in documentation, see the chapter "Documenting Your Data Model"

Change a File Name

Follow these steps:

1. On the Edit Database Relations panel, zoom into the file details by typing Z against any of the relations of the file.
2. On the Edit File Details panel, press F8**,** type the file name, and press Enter.

Change Enhance SQL Naming

Follow these steps:

1. On the Edit Database Relations panel, zoom into the file details by typing Z against any of the relations of the file.
2. On the Edit File Details panel, press F8=Change name**,** change the value for the Enhance SQL Naming option to Y or N, and press Enter.

   Note: Default value=N.

Delete a File

To delete a file, you must delete the Defined as relation for that file. Before you can delete a Defined as relation, you must first delete all other references to the file. Use the positioning option to view all references.

Follow these steps:

1. Delete all relations, except a Defined as relation.

   This includes all Owned by, Known by, Refers to, Has, Extended by, Qualified by, and Includes relations.

   On the Edit Database Relations panel, type a D next to each of the relations of the file and press Enter.

2. Delete the Defined as relation (DFN).

   When you delete the Defined as relation, CA 2E removes all the access paths and functions associated with this file.

   - On the Edit Database Relations panel, enter DFN for the Rel lvl and press Enter.
   - Type a D next to the Defined as relation and press Enter.

3. Optionally delete all fields that are now unreferenced.

   From the Edit Database Relations panel:

   - Press F7 to display the Display Fields panel.
   - Press F11 to display unreferenced fields.
   - Enter a D next to each of the fields to be deleted and press Enter.

4. Delete all messages for this file. This includes all user-created messages in addition to the default messages.

   There are two default messages:

   "File name" EX (record already exists)

   "File name" NF (record not found)

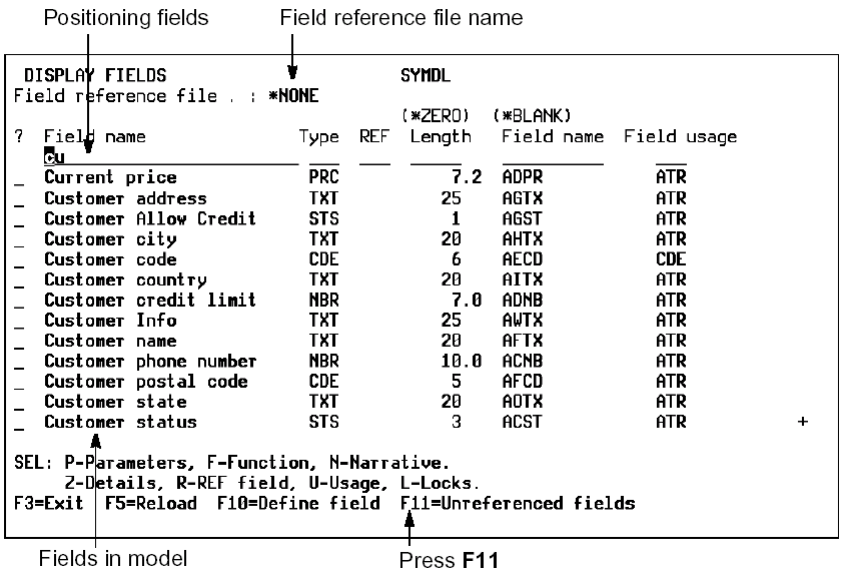
   - On the Edit Database Relations panel, type an *****m above the Object field to position to the *Messages file. Press Enter.
   - Type F against any Message field and press Enter.
   - On the Edit Message Functions panel, type D next to the message to be deleted and press Enter.

Fields

Delete a Field

Follow these steps:

1. From the Edit Database Relations panel, press F7 to display the Display Fields panel.



2. A field can be deleted only if it is not referenced by any other field, file, or relation. Press F11 from the Display Fields panel to display a list of unreferenced fields in your model.
3. Type D next to each of the fields to be deleted and press Enter.

Conditions

Delete a Condition

A condition cannot be deleted if it is referenced. You must first remove all the references where the condition is used before you can delete a condition. A condition may have been used in other places such as in an access path or action diagram.

On the Edit Field Conditions panel, type D next to the condition you want to delete and press Enter.

Relations

Add Narrative Text

Follow these steps:

1. On the Edit Database Relations panel, type N0 next to a relation. The Edit Narrative Text panel displays.
2. Fill this panel with the text, notes, or any description you want to include. Press Enter, F3 to exit.

For more information:

- On using narrative text, see the *Administration Guide*.
- On how to include narrative in documentation, see the chapter "Documenting Your Data Model"

Change a Relation

You can change a relation by typing over that relation's statement on the Edit Database Relations panel. The file (object), the relation, and the related file or field (referenced object) on the statement can be changed. CA 2E provides automatic relation syntax checking to prevent entry of invalid statements.

Follow these steps:

1. On the Edit Database Relations panel, type a new object, relation, or referenced object (file or field) over the existing name and press Enter.
2. Press F10 to define objects if the referenced object is not yet defined.

Note: To change the file or field name, do not type over the name and define a new object. Zoom into the file or field details and change the name.

Override Default Relations Sequence

You can change the sequence order of relations by entering a new sequence number using the Sequence field on the relation statements.

For more information on sequencing redirected or shared relations, see the chapter "Creating/Defining Your Data Model."

On the Edit Database Relations panel, type the new sequence number(s) in the Relation Seq column and press Enter.

Delete a Relation

To delete a relation, type D next to the relation to be deleted and press Enter.

Note: Defined as relations can only be deleted if there are no references by other relations to the file it defines.

Example: File B cannot be deleted, that is the Defined as relation cannot be deleted, until the other relations have been deleted. This includes relations that reference that file. If A refers to B, the file B cannot be deleted until the relation A Refers to B is removed.

FIL	B	REF	Defined as	FIL	B	REF
FIL	B	REF	Known by	FLD	b1	REF

and also,

FIL	A	REF	Refers to	FIL	B	REF

Deleting the Defined as relation removes all functions and access paths built over this file, including default functions and access paths.

Creating User-Defined Field Types

CA 2E provides a wide range of default field types that cover many data requirements. If you need a field type that is not defined within the CA 2E product, you can create your own. This topic provides instructions on defining and creating your own field types in addition to the ones supplied by CA 2E's shipped file *Field Attribute Types.

Included in this topic are examples for creating the following user-defined field types.

- Century Date Field Type
- Currency Field Type
- Real Percentage Field Type

Each specific CA 2E user-defined field type has a number of attribute values that will be assigned to any new fields given that type. You can specify which of those values cannot be changed and which can be overridden at the individual field level. You may also specify value mapping and validation routines centrally to the data model for your new field type.

Note: Only a user of type *DSNR may define new field types.

Name and Text

You must specify a 3-character mnemonic for the field type you define. You cannot duplicate existing field type mnemonics.

You can also add explanatory text to a field. The text will appear next to the field name shown on the Display Object Attribute panel.

You should define a unique 2-character mnemonic code, which will be used to generate field names for fields of the same field type. The field name mnemonic will be defaulted to the first two letters of the 3-character field type.

Basic Attributes

You can specify default values for all basic field attributes. These are

- i OS data type
- Internal and external length
- Keyboard shift character
- Allow lowercase
- Check valid system name
- Mandatory fill required
- Modulus 10/11 check
- Values mapping

For more information refer to the Edit Field Type panel and the description of attributes that follows.

Internal and External Length

You can specify default values for both the internal and the external length of the field; these lengths may be different. You can either specify fixed values or allow the user to supply both internal and external values. You can also calculate the external length from the supplied internal length with a length conversion function.

For more information on field length calculation, see the examples following.

Mapping Functions

If a field is to be stored internally in a form different from the one in which it is displayed externally, then you can specify mapping functions to describe how the values are to be translated.

These functions can be any CA 2E functions. They must be attached to the CA 2E shipped file *Field attribute types. The functions must have at least one parameter. There are restrictions about how other parameters can be specified.

For each field, you can use mapping functions in each direction as described below.

- External-to-Internal function: to specify how the entered value for the field is to be mapped to the internal value. Validation may be included in this function.
- Internal-to-External function: to specify how the stored value is to be converted to the displayed value. You must specify value mapping for the field type if you specify an internal-to-external function. To specify value mapping, enter a **Y** in the Initial value column for Allow value mapping.

Note: If you use an EXCUSRSRC function, only define your work variables once, in the Internal-to- External function.

For more information on mapping functions, see Specifying Mapping Functions.

Defining New Field Types

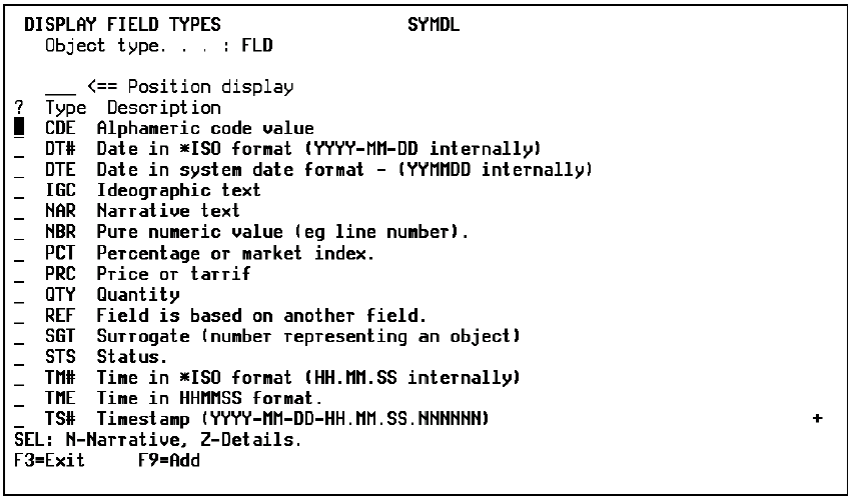
You use the Edit Field Type panel to define new field types. This panel enables you to update the attributes for CA 2E field user-defined types. This panel is display only for CA 2E system field types.

Follow these instructions to define your own field type:

1. On the Edit Database Relations panel, type *F in the Object positioner area to display the *Field attribute types file.

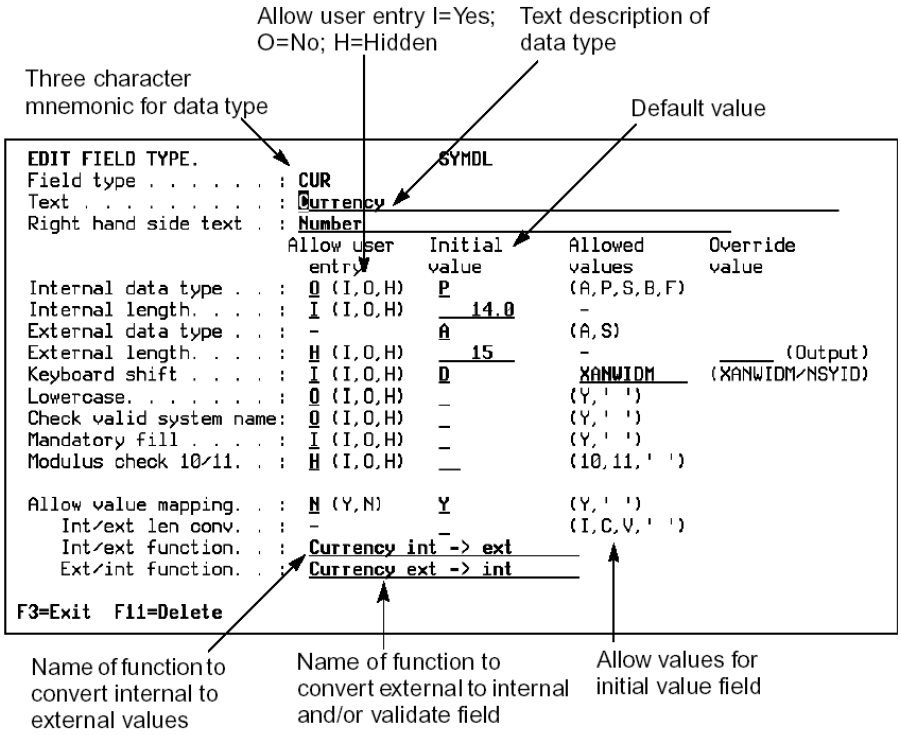
   Type Z against one of the relations for the *Field attribute types file to access the field types. The Display Field Types panel displays.

   This panel shows a list and the description of all the existing field types contained in the shipped file.



2. Press F9 to display the Edit Field Type panel, which is where you define a new field type. Enter all of the necessary information to specify the characteristics of the field type. The options are explained following the panel.

Edit Field Type Panel



Specifying Basic Attribute Values

This panel shows the attributes of the field type (DTX, as used for the first example) and highlights the two main columns where you can change its values.

The first column, Allow user entry, indicates how the field type attributes will be displayed when a field of this type is shown for editing on the Edit Field Details panel; the second column, Initial values, is for specifying the initial default values allowed for that field type.

For example, you would select and change the values for the Internal data type of the DTX attribute as follows:

Attribute	Allowed user entry	Initial value
Internal data type	O (I,O,H)	P

For Allow user entry, O means the Internal data type attribute is to be displayed with a fixed value; I means it can be modified; and H means it is to be hidden altogether.

For Initial value, P means you want Internal data type to have the value of Packed numeric.

The following description of the attributes and default values shown for a field type on the Edit Field Type panel will explain further how you define a new field type.

- **Field type—**The field type for which the attribute values are to be specified.
- **Text—**A short description of the field type, for example, "8-character date field."
- **Right hand side text—**the default right hand side text for fields of this type.
- **Internal data type—**The system data type of the field to be stored on a database file. It can have one of these values:

A:	Alphanumeric
P:	Packed numeric
S:	Signed numeric (zoned)
B:	Binary (does not get generated)
F:	Floating (does not get generated)

- **Internal length—**The number of bytes used to store a field in files. Fields with decimal positions are entered as: total number of digits, number of decimal places. For example, for a field to contain 999.99, the length would be 5.2.
- **External data type—**System data type of field to be displayed on device files. It can have one of these values:
-

A:	Alphanumeric
S:	Signed numeric (zoned)

- **External length—**The number of characters or digits in a field on a panel or print file. Fields with decimal positions are entered as: total number of digits, number of decimal places. For example, for a field to contain 999.99, the length would be 5.2.
- **Keyboard shift—**Specifies which keyboard shift is allowed for the field on panel files. It can have one of these values:

Blank	no keyboard shift
X, A, N, W, I, D, or M:	alphanumeric fields
N, S, Y, I, or D:	numeric fields
O, J, E, W, G, or A:	ideographic fields

For more information on keyboard shift values, refer to the IBM *DDS Reference* manual.

- **Lowercase**—specifies whether the field values may be in lowercase. Lowercase applies only to alphanumeric fields. It can have one of these values:

Y:	lowercase allowed
Blank:	lowercase not allowed

- **Check valid system name**—specifies whether any value entered for the field must be a valid i OS system name. A valid system name must start with a letter, no more than 10 characters long, and must contain only letters, digits or one of these characters: "-", "#", "$", or "@".
- **Mandatory fill**—specifies whether an entry if any must be made for every character of the field. This can have one of these values:

Y:	mandatory fill
Blank:	no mandatory fill

- **Modulus check 10/11**—specifies whether any value entered for the field must meet a modulus 10 or 11 check as specified by the DDS CHECK keyword. Modulus check applies only to numeric fields. This can have one of these values:

10:	apply modulus 10 check
11:	apply modulus 11 check
Blank:	do not apply modulus check

- **Allow value mapping**—specifies whether value mapping will be implemented for the field. If value mapping is 'Y', you must specify an Int/ext function and an Ext/int function. If no value mapping is required, you can still specify an Ext/int function to perform other validation.

This can have one of these values:

Y:	field is to be value mapped
Blank:	field is not to be value mapped

- **Int/ext len conv**—specifies how external length is to be determined. This can have one of these values:

I:	use internal length. This will force the External length value to equal the Internal length value on the Edit Field Type screen.

C:	invoke user program. The calculation will be performed by a field length calculation program named YxxxLENR1C where "xxx" is the 3-character name of the data type. You must supply this program yourself.
V:	if the field is valued mapped, allow user entry of the external length. If it is not value mapped, use internal length.
Blank:	allow user entry of the external length.

- **Int/ext function**—This is the function that is to convert the internal value to external value for the field. It must be a function attached to the *Field attribute types file.
- **Ext/int function**—This is the function that is to convert the external value to internal value for the field and validate it if required. It must be a function attached to the *Field attribute types file.

Specifying Mapping Functions

You can examine and change the value mapping functions in a model using the Edit Functions panel for the *Field attribute types file. This panel shows the mapping functions attached to the field types file.

You can obtain the Edit Functions panel in two ways:

- Type **?** for the Ext/int function or the Int/ext function fields on the Edit Field Type panel.

  This will show you a list of functions for the *Field attribute types. Functions can be added on the Edit Functions panel.

- Type F next to a relation of the *Field attribute types on the Edit Database Relations panel.

When you define a function that is based on an access path containing fields of a user-defined field type, CA 2E automatically does the following in the function definition:

- Include fields of the external field type on the device designs of the function.
- Include the field mapping functions at the appropriate points in the action diagrams.
- Pass the internal and external fields to the mapping function parameters.
- Specify that the internal fields are to be used to update the database.

If value mapping is specified for a field type (in other words, **Y** is specified in the Initial value column of Allow value mapping), both internal/external and external/internal functions must be defined and both must have two parameters. This is to supply and return the internal and external values as appropriate.

If the external/internal function is being used for validation only, only one parameter needs be supplied: the external value.

Function type	First parameter	Parameter Usage	Second Parameter	Parameter Usage
External/Internal	EXT FLD	I	INT FLD	O
Internal/External	EXT FLD	O	INT FLD	I

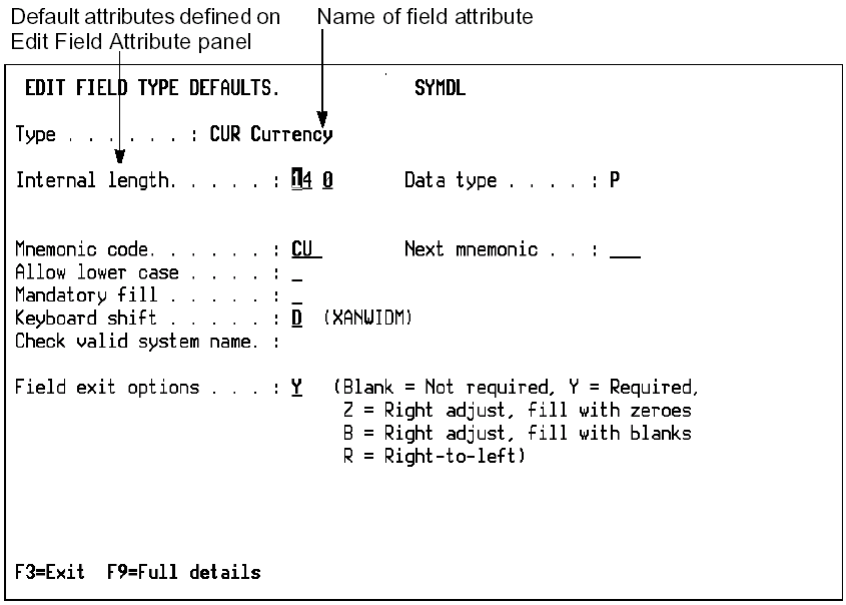
The order in which you specify the parameters is important. Ensure the following:

- The external version of the field must always be the first parameter, regardless of usage. The field used to define the external parameter should have a field type of NBR, CDE or TXT.
- The field used to define the internal parameter should always be the second parameter. You can specify additional parameters after these two fields.

Specifying Additional Attribute Values

Use the Edit Field Type Defaults to specify the default initial values for some additional basic attributes of the field; for instance, field exit options and edit codes. You can obtain this panel in either of two ways:

- For existing field types, enter Z next to the field type on the Display Field Types panel
- Press Enter when defining a new attribute type from the Edit Field Type panel.



The Edit Field Type Defaults panel allows you to enter and change the following attributes of the field type:

- Internal length
- External length
- Data type
- Mnemonic code
- Keyboard shift
- Lowercase
- Mandatory fill
- Modulus 10/11
- Valid System Name (VNM)
- Field Exit options
- Edit codes

Example: Defining a Century date Field Type (DTX)

This example shows how to define a century date field data type (DTX). Since the shipped DT# field type has the capabilities defined here, use this example only to understand the steps needed to define your own field type. The DTX field type is to have the following characteristics.

- Internal format packed numeric: CCYYMMDD, fixed length
- External display format numeric: DD/MM/CCYY, fixed length
- Value mapping to convert between the internal and external values
- Validation to check that entered dates are valid

The first step is to define the mapping functions needed for the DTX field type.

Check DTX function provides external-to-internal conversion for DTX data type

```
 EDIT FUNCTIONS                          SYMDL
File name. . . . : *Field attribute types                    **   1ST LEVEL  **

? Function                Function type                Access path
P  Check DTX              Execute internal function *NONE
_ Check DT#               Execute internal function *NONE
_ Check DTE               Execute internal function *NONE
_ Check STS               Execute internal function *NONE
_ Check TM#               Execute internal function *NONE
_ Check TME               Execute internal function *NONE
_ Check TS#               Execute internal function *NONE
_ Convert DT#             Execute internal function *NONE
_ Convert DTE             Execute internal function *NONE
P Convert DTX             Execute internal function *NONE
_ Convert STS             Execute internal function *NONE
_ Convert TM#             Execute internal function *NONE
_ Convert TS#             Execute internal function *NONE
                                                                        More...
SEL: Z-Details, P-Parms, F-Action diagram, S-Device design, N-Narr, O-Open,
  T-Structure, A-Access path, U-Usage, G/J-Generate, D-Delete, C-Copy, L-Lock.
F3=Exit        F5=Reload                  F7=File details   F9=Add function
F17=Services   F21=Copy *Template function
```

Convert DTX function provides internal-to-external conversion for DTX data

Type P next to the Check DTX and Convert DTX to display their parameters. Press Enter.

Defining Parameters for the Mapping Functions

You use the Edit Function Parameter and Edit Function Parameter Detail panels to specify the parameters for mapping functions.

The two mapping functions for the DTX field data type, Check DTX and Convert DTX, should have parameters defined for them as follows:

Function Name		FIRST PARAMETER			SECOND PARAMETER		
	Function Type	Name	Usage	Data Type	Name	Usage	Data Type
Check DTX	Ext/Int	Century external	I	NBR 8.0	Century internal	O	
Convert DTX	Int/Ext	Century external	I			O	DTX 8.0
			O	NBR 8.0	Century internal	I	DTX 8.0

The fields Century internal and Century external are used only to define the parameters on the century data type mapping functions. They can be defined with the Define Objects panel and should be of the same data type (packed, zoned, alphanumeric, etc.) and size as the respective internal and external formats of the field data type. For the DTX field, both internal and external formats are numeric. The Century internal field could itself be a field of type DTX.

If necessary, use the Define Objects panel to define fields of appropriate data types to use as parameter fields on the mapping functions.

Defining the Mapping Functions

Having specified the parameters for the Check DTX and Convert DTX functions, you can describe the internal processing of the functions themselves, using an action diagram of type Execute Internal Function. The Check DTX function would do the following:

1. Check that the date entered is a valid date of the form DDMMCCYY.
2. Convert the external value of the DTX field, in DDMMCCYY format, into the internal value, in CCYYMMDD format.

The Convert DTX function has only to convert the internal value of the field, in CCYYMMDD format, to the external value, in DDMMCCYY format.

Supplying Parameters to Mapping Functions

When a function uses a field with a mapped user-defined field type, calls to the mapping functions are included automatically. The internal and external values are automatically supplied to the required parameters of the mapping functions.

If you specify additional parameters on a mapping function, you must also decide from where the values for those parameters are to be obtained in the functions, which use the field type.

For each parameter you can specify:

- The name of another field present in the same format as the field of the data type (FIL).
- A condition value of a field in the same format as the field of the data type (CND).
- The name of a field in the JOB context.
- A constant value (CON).

The source of field mapping function parameters can be specified at two levels:

- File entry
- Function screen/report entry

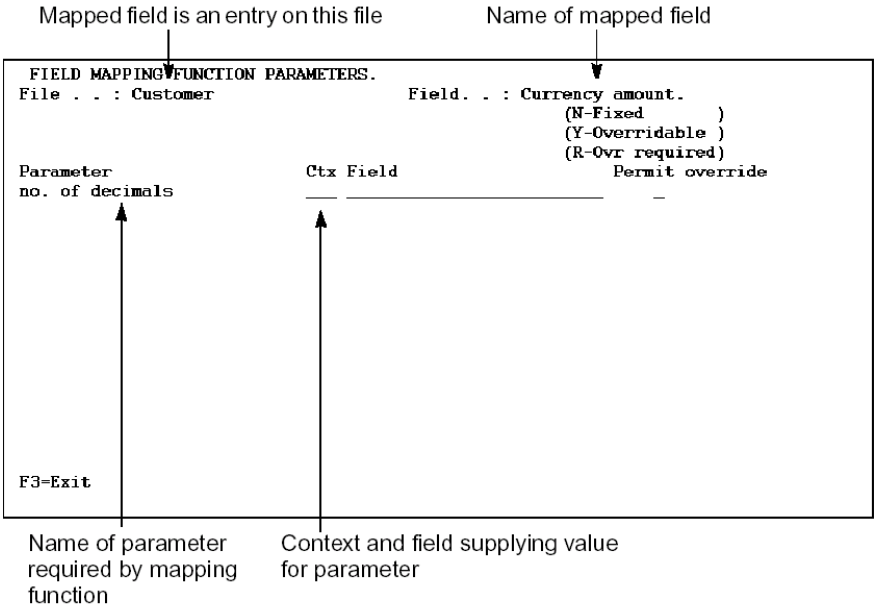
  You can also specify whether the values specified at file level may be overridden at a device design level.

To specify the context of the values for any additional parameters on the mapping functions, you use the Field Mapping Function Parameters and the Screen Field Mapping Function Parameters panels.

The Field Mapping Function Parameters panel allows you to specify a default context for the mapping parameters.

The example of the Field Mapping Function Parameters panel below shows how to define the No. of decimals parameter for mapping fields of data type CUR described earlier under "Specifying Additional Parameters for Mapping Functions." Note that you can override the source of the parameter at a lower level.

Field Mapping Function Parameters Panel



To access this panel from the Edit Database Relations panel, use an E to obtain the Edit File Entries panel to display the entries of the file that has a function containing the mapped user-defined field. Use an M against the entry to set up the additional parameter to the mapped user-defined field type function.

For more information see Displaying File Entries at the beginning of this chapter.

For each mapping function parameter you can enter the context and the name of the field which is to supply the value to the mapping function parameters. This value will be used in all functions based on the file unless it is overridden at the device level.

A special context is available on this panel: FIL. This context indicates that the parameter field is to be found in the same context as the mapped field, wherever it is being used, such as DTL or RCD on a screen, CUR on a report.

You can also specify whether the source of the mapping function parameter is fixed or whether it can be overridden at the panel or report level. The third alternative is to specify that an override is always required.

Specifying Additional Parameters for Mapping Functions

You can specify additional parameters on the mapping functions if you wish. For example, for the currency amount data type you might specify that the number of decimal places was an additional input parameter to the mapping functions. This parameter could then be used to control the positioning of the decimal place.

Mapping Function Parameters: Panel/Report Entry Level

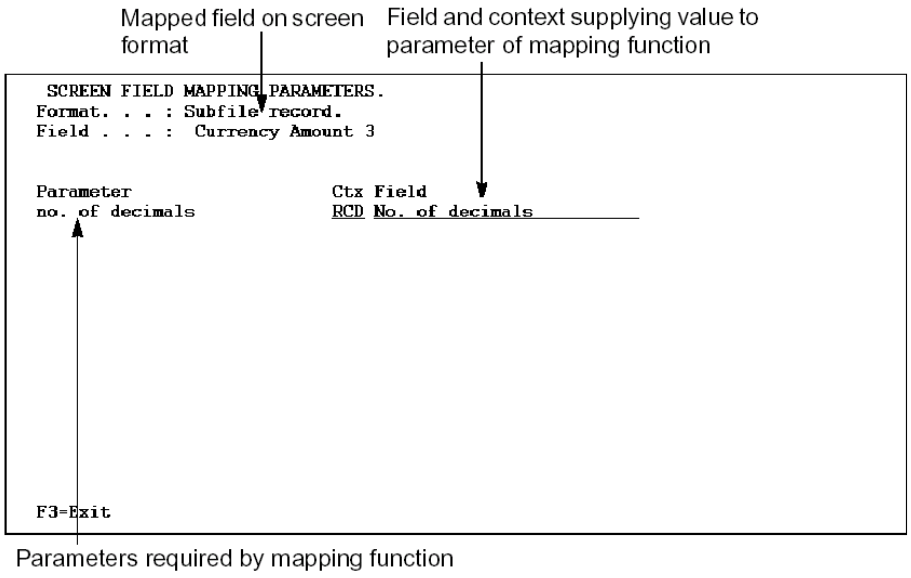
To override the parameters to a mapping function at panel or report level, you use the Screen Field Mapping Parameters panel.

1. From the Edit Screen Entry Details panel, press F9 to obtain the Screen Field Mapping Parameters panel.
2. From the device design editor for a function panel, use F5 to get to the Edit Device Format panel. From here, Z (Zoom) into the panel entry.

This will take you to the Edit Screen Entry Details panel.

For a field of the currency data type, the No. of decimals parameter could be overridden since the parameter value was not protected at the file level. Note that the context FIL, specified at the file entry level, is resolved into an appropriate screen context; for example, for an EDTFIL function, RCD is the screen context.

Screen Field Mapping Parameters Panel



Example: Defining a Currency Field Type (CUR)

User-defined field types are useful where data requires some conversion before being displayed, or where a standard conversion is required prior to storage.

The conversion process does not have to be two-way. For example, a user-defined type could be used to allow an automobile registration number to be processed to convert all zeros to the letter O and all ones to the letter I to ensure against confusion on inquiry screens. The original data could be stored in another field if necessary.

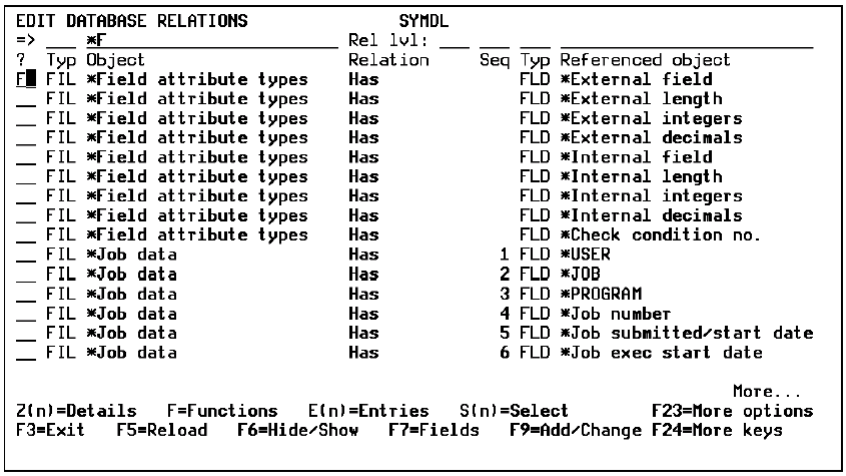
In the automobile registration number example there is no conversion from internal to external format since they are both the same. The only parameter to the external to internal conversion function is the field to be converted.

The following example is somewhat more complicated to illustrate the power of user-defined field types. In it we will define a currency field type. A company dealing with international customers who are charged and who pay in their own currency would want to define a value field only once for a particular data item, rather than various fields or even separate files to store the data.

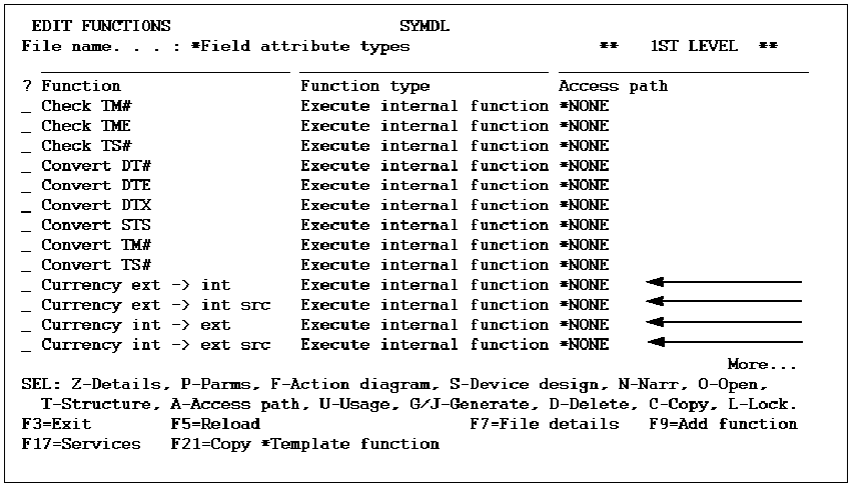
The solution is to store the data in a neutral format and then convert it before display with the correct number of decimal places and any other desired editing. Rather than require the developers to remember to call these conversion routines, a user-defined data type allows the definition to be made once and is then automatically generated by CA 2E when required.

The following panels describe the definition process step by step.

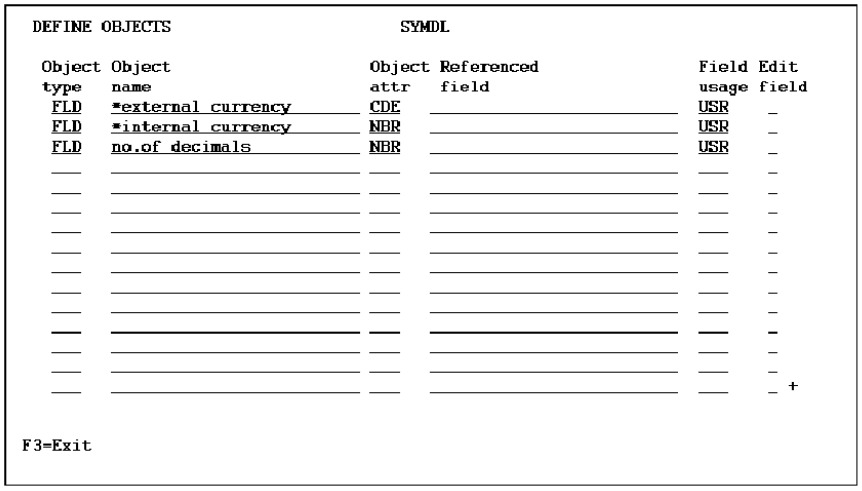
1. First define the functions used for the field type on the *Field attribute types file. From the Edit Database Relations panel, enter *****F in the object positioner field to position on the CA 2E *Field attribute types file. Enter F next to the *Field attribute types file name to display the functions that have been defined for this file.



2. Create the four following mapping functions that will be used by the currency field type as shown below. The access path for each function type should be *NONE.

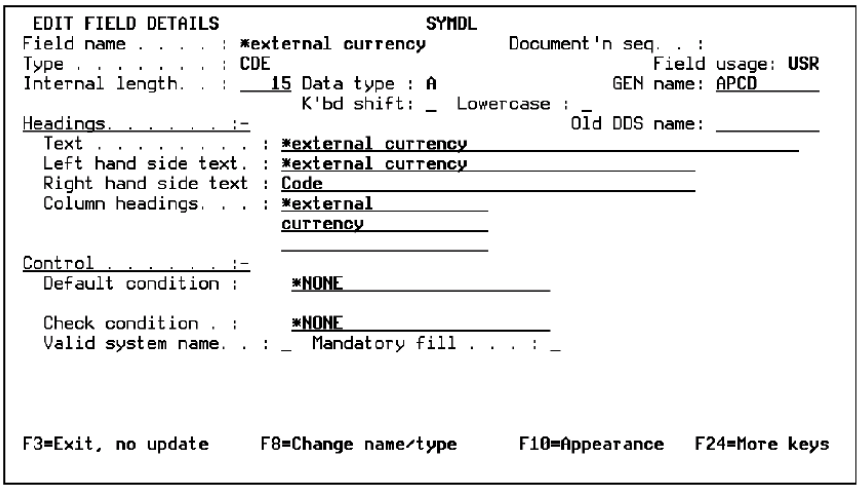


3. For the purposes of this example, the following new fields are required. You can create them as part of the function creation process.

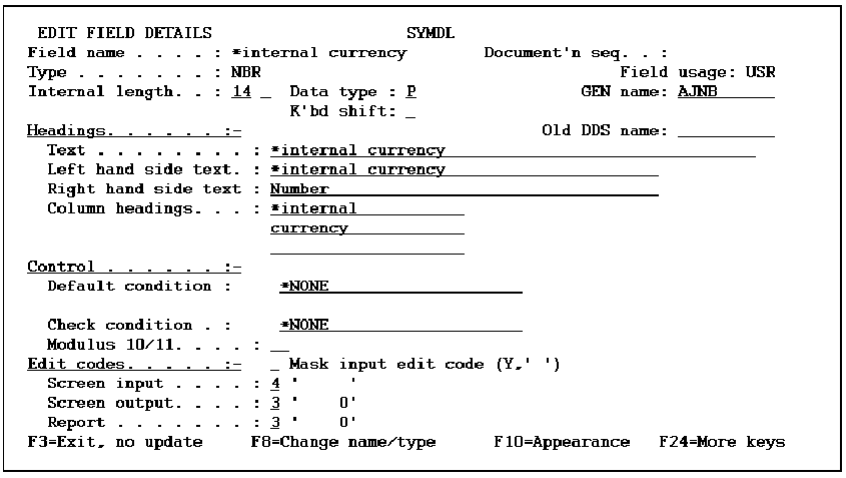


4. Press Enter to define the fields and return to the Display Fields panel. Zoom into each field to display the Edit Field Details panel.

   Enter details for the *external currency field as follows.



   Enter details for the *internal currency field as follows.



Enter details for the no. of decimals field as follows.

```
 EDIT FIELD DETAILS                    SYMDL
Field name . . . . : *no. of decimals        Document'n seq. . :
Type . . . . . : NBR                                Field usage: USR
Internal length. . : 2 _  Data type : P            GEN name: ANNB
                          K'bd shift: _
Headings. . . . . :-                          Old DDS name:
  Text . . . . . : *no. of decimals
  Left hand side text. : *no. of decimals
  Right hand side text : Number
  Column headings. . . : *no. of
                         decimals

Control . . . . . :-
  Default condition :    *NONE

  Check condition . :    *NONE
  Modulus 10/11. . . . : _
Edit codes. . . . . :-   _ Mask input edit code (Y,' ')
  Screen input . . . . : 4 '  '
  Screen output. . . . : 3 ' 0'
  Report . . . . . : 3 ' 0'
F3=Exit, no update     F8=Change name/type      F10=Appearance   F24=More keys
```

Note the assigned DDS field names for the fields you just defined. In this case, XBCD, ZZNB, and QHNB. You will need these when you define the user source for the ext –> int src and the int –> ext src functions.

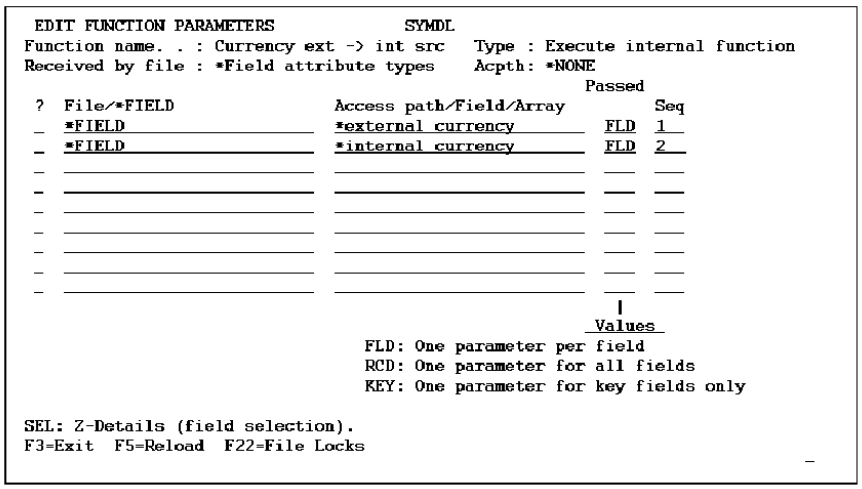
5. Define parameters to the Currency ext –> int EXCINTFUN function as shown below using the fields you just defined. Be sure to define the *external currency field first. You can use a sequence number to ensure this.

```
 EDIT FUNCTION PARAMETERS              SYMDL
Function name. . : Currency ext -> int      Type : Execute internal function
Received by file : *Field attribute types   Acpth: *NONE
                                                 Passed
 ?  File/*FIELD              Access path/Field/Array        Seq
 _  *FIELD                   *external currency       FLD  1
 _  *FIELD                   *internal currency       FLD  2
 _  _____________________    ____________________     ___  ___
 _  _____________________    ____________________     ___  ___
 _  _____________________    ____________________     ___  ___
 _  _____________________    ____________________     ___  ___
 _  _____________________    ____________________     ___  ___
 _  _____________________    ____________________     ___  ___
 _  _____________________    ____________________     ___  ___
                                                      |
                                                    Values
                                  FLD: One parameter per field
                                  RCD: One parameter for all fields
                                  KEY: One parameter for key fields only

SEL: Z-Details (field selection).
F3=Exit  F5=Reload  F22=File Locks
```

Zoom into each parameter to display the Edit Function Parameter Details panel. Assign the usage and role for the Currency ext –> int function parameters as follows.

Parameter	Usage	Role
*external currency	I	MAP
*internal currency	O	MAP

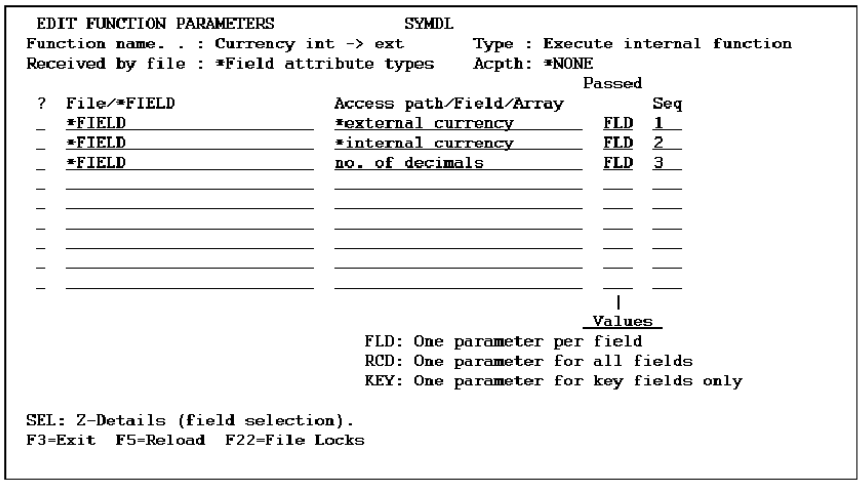
6. Define parameters to the Currency ext –> int src EXCUSRSRC function as shown below. Be sure to define the *external currency field first. You can use a sequence number to ensure this.



Zoom into each parameter to display the Edit Function Parameter Details panel. Assign the usage and role for the Currency ext –> int src function parameters as follows.

Parameter	Usage	Role
*external currency	I	MAP
*internal currency	O	MAP

7. Define parameters to the Currency int –> ext EXCINTFUN function as shown below. Be sure to define the *external currency field first. You can use a sequence number to ensure this.



Zoom into each parameter to display the Edit Function Parameter Details panel. Assign the usage and role for the Currency int –> ext function parameters as follows.

Parameter	Usage	Role
*external currency	O	MAP
*internal currency	I	MAP
no. of decimals	I	MAP

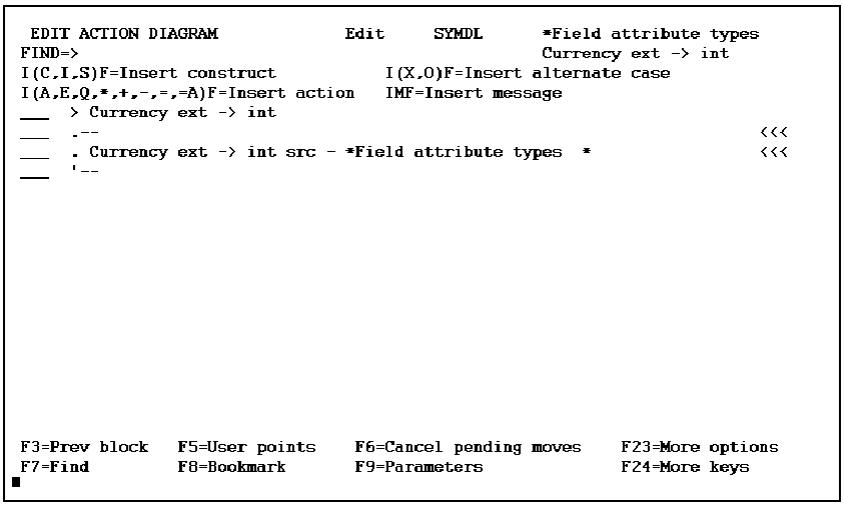
8. Define parameters to the Currency int –> ext src EXCUSRSRC function as shown. Be sure to define the *external currency field first. You can use a sequence number to ensure this.

```
 EDIT FUNCTION PARAMETERS                SYMDL
Function name. . : Currency int -> ext src   Type : Execute internal function
Received by file : *Field attribute types    Acpth: *NONE
                                                 Passed
 ?  File/*FIELD                Access path/Field/Array          Seq
 _  *FIELD                     *external currency          FLD  1
 _  *FIELD                     *internal currency          FLD  2
 _  *FIELD                     no. of decimals             FLD  3
 _  ____________________       ____________________        ___  ___
 _  ____________________       ____________________        ___  ___
 _  ____________________       ____________________        ___  ___
 _  ____________________       ____________________        ___  ___
 _  ____________________       ____________________        ___  ___
 _  ____________________       ____________________        ___  ___
                                                    |
                                                  Values
                                  FLD: One parameter per field
                                  RCD: One parameter for all fields
                                  KEY: One parameter for key fields only

SEL: Z-Details (field selection).
F3=Exit  F5=Reload  F22=File Locks
```

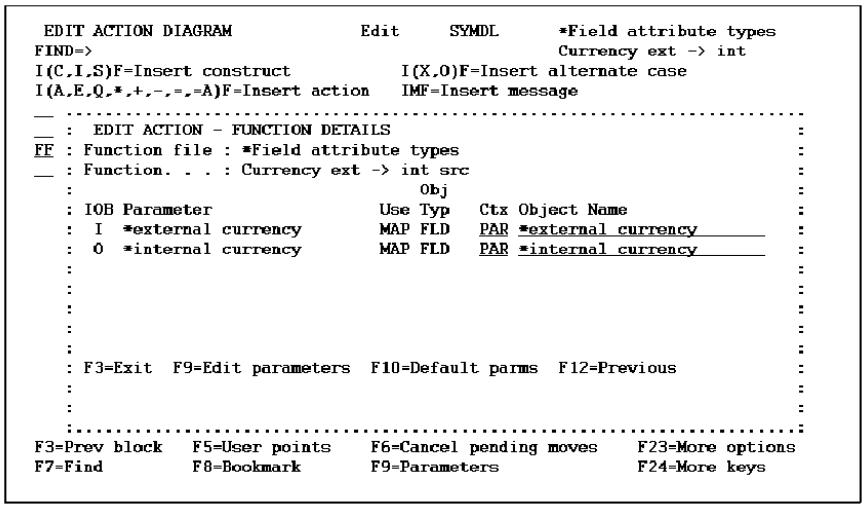
Zoom into each parameter to display the Edit Function Parameter Details panel. Assign the usage and role for the Currency int –> ext src function parameters as follows.

Parameter	Usage	Role
*external currency	O	MAP
*internal currency	I	MAP
no. of decimals	I	MAP

9. Edit the action diagram for the Currency ext –> int EXCINTFUN to call the EXCUSRSRC function for the external to internal conversion.



10. Specify the details of the parameter interface.

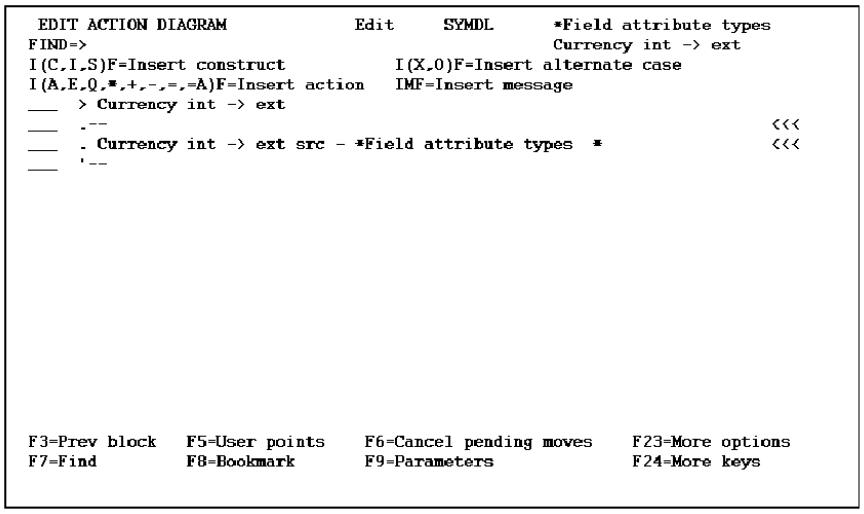


11. The RPG source for the Currency ext –> int src EXCUSRSRC function is as follows. The process involves the removal of any decimal point found in the data. Note that to reduce the complexity of the example no validation has been included.

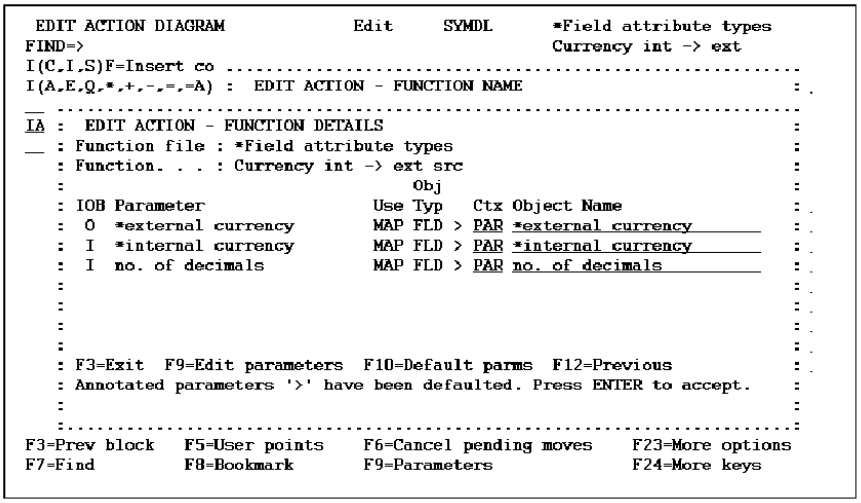
```
*CURRENCY EXT –> INT SRC

                                        *CONVERT TO INTERNAL FORMAT
```

	C C C	* *	Z-ADD15 Z-ADD14 MOVEA	#IXBCD	X Y OUT	
		*STRIP OUT DECIMAL POINT				
	C C C C C C C C C	X OUT,X *	DOUEQ1 IFNE MOVE SUB END SUB END MOVEA MOV	'.' OUT,X 1 1 INP WRK14	INP,Y Y X WRK14 #OZZNB	

1. Edit the action diagram for the Currency int –> ext EXCINTFUN to call the EXCUSRSRC function for the internal to external conversion.



2. Specify the details of the parameter interface.



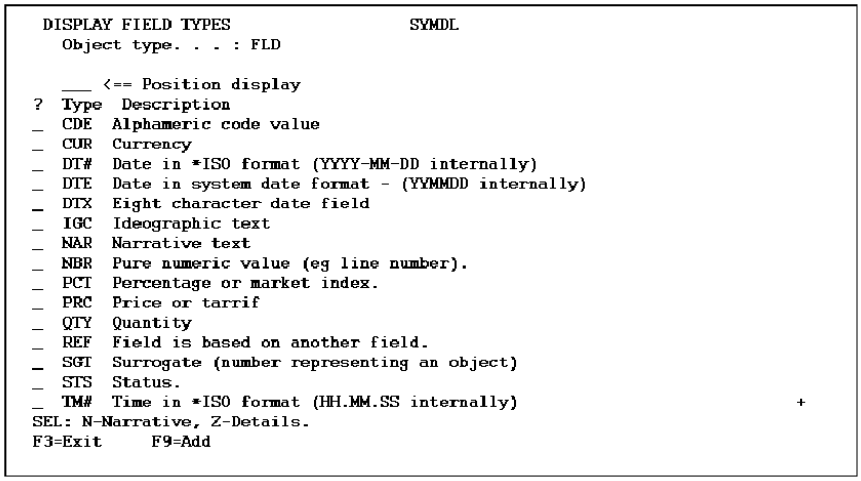
3. Following is the RPG source for the Currency int –> ext src EXCUSRSRC function.

*CURRENCY INT –> EXT SRC

```
     E     *              INP           14    1
     E     *              OUT           15    1
     C           #IQHNB   MOVE          ' '   OUT
     C     *              MOVE          #IZZNB WRK14
     C           #IQHNB   Z-ADD         #IQHNB WRK2N
     C                    MOVEA         WRK14 INP
     C                    MOVEA         INP   OUT
     C                    IFEQ          1     OUT,14
     C                    MOVE          '.'   OUT,15
     C                    MOVEA         INP,14 OUT,13
     C                    ELSE          2     OUT,14
     C                    IFEQ          '.'   OUT,15
     C                    MOVE          INP,13
     C                    MOVEA         INP,14
     C                    MOVEA
     C                    ELSE
     C           #IQHNB   IFEQ          3     OUT,12
     C           #IQHNB   MOVE          '.'   OUT,13
     C                    MOVEA         INP,12 OUT,14
     C                    MOVEA         INP,13 OUT,15
     C                    MOVEA         INP,14 OUT,11
     C                    ELSE          4     OUT,12
     C                    IFEQ          '.'   OUT,13
     C                    MOVE          INP,11 OUT,14
     C                    MOVEA         INP,12 OUT,15
     C                    MOVEA         INP,13
     C                    MOVEA         INP,14
     C                    MOVEA
     C                    END
     C                    END
     C                    END
     C                    END
```

		*CHANGE LEADING ZEROES TO BLANKS				
	C C C C C C	OUT,X * * 	Z-ADD1 DOUNE MOVE ADD END MOVEA	'0' ' ' 1 OUT	X OUT,X X #OXBCD	

1. Zoom into the *Field attribute types file from the Edit Database Relations panel to display a list of the currently defined field types.



2. Press F9 from the Display Field Types panel to add the new field type CUR. Specify attributes for the CUR field type on the Edit Field Type panel as follows. Be sure to enter Y for the initial Allow value mapping value.

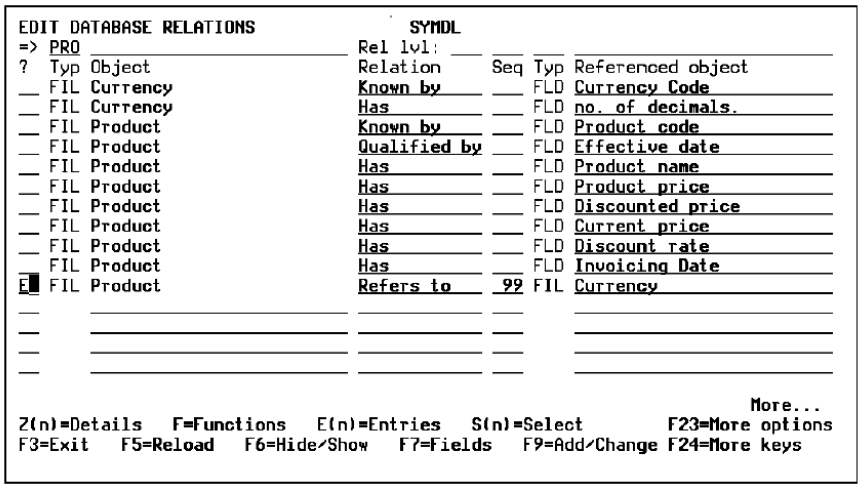
```
EDIT FIELD TYPE.                          SYMDL
Field type . . . . . : CUR
Text . . . . . : Currency
Right hand side text . : number
                        Allow user    Initial      Allowed      Override
                          entry       value        values       value
Internal data type . . : O (I,O,H)    P            (A,P,S,B,F)
Internal length. . . . : I (I,O,H)      14.0       -
External data type . . : -            A            (A,S)
External length. . . . : H (I,O,H)      15         -             _____ (Output)
Keyboard shift . . . . : I (I,O,H)    D            XANWIDM       (XANWIDM/NSYID)
Lowercase. . . . . : O (I,O,H)        _            (Y,' ')
Check valid system name: O (I,O,H)    _            (Y,' ')
Mandatory fill . . . . : I (I,O,H)    _            (Y,' ')
Modulus check 10/11. . : H (I,O,H)    __           (10,11,' ')

Allow value mapping. . : N (Y,N)      Y            (Y,' ')
   Int/ext len conv. . : -            _            (I,C,V,' ')
   Int/ext function. . : Currency int -> ext
   Ext/int function. . : Currency ext -> int

F3=Exit
```

3. Press Enter to confirm the values you entered. Press F3 to specify additional attributes for the CUR field type on the Edit Field Type Defaults panel as follows.

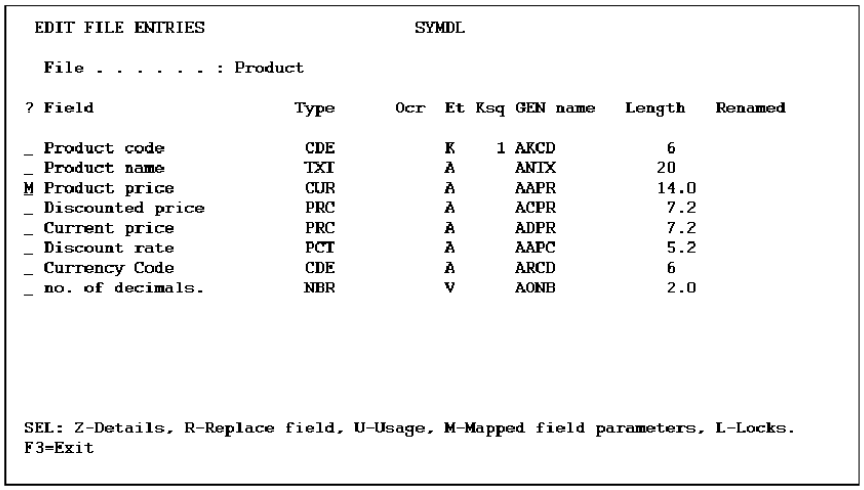
```
 EDIT FIELD TYPE DEFAULTS.                SYMDL

 Type . . . . . : CUR Currency

 Internal length. . . . . : 14 0       Data type . . . . : P


 Mnemonic code. . . . . : CU           Next mnemonic . . : ___
 Allow lower case . . . . : _
 Mandatory fill . . . . . : _
 Keyboard shift . . . . . : D  (XANWIDM)
 Check valid system name. :

 Field exit options . . . : B   (Blank = Not required, Y = Required,
                                 Z = Right adjust, fill with zeroes
                                 B = Right adjust, fill with blanks
                                 R = Right-to-left)



 F3=Exit  F9=Full details
```

4. In order to use the CUR field type, you need a Currency file as shown below. The records on this file will reflect the number of decimal places required by the various currency types.

   Any files that actually use the CUR field type must reference the Currency file. The no. of decimals field must be virtualized across the Refers to relation.

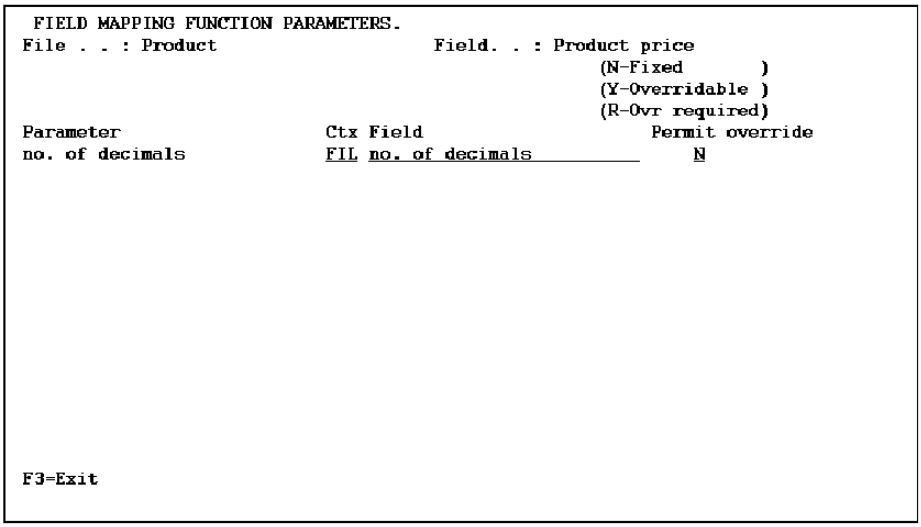


5. In order to map the value of the no. of decimals to the field type, display the entries on the file by entering E on the Edit Database Relations panel as shown above. Enter M against the entry defined using the currency field type to define the additional parameter to the mapped user-defined field type function.



6. This panel allows the designer to define from which context the actual parameters may be selected.

For more information on this process, refer to this topic, the Supplying Parameters to Mapping Functions subtopic, the Mapping Function Parameters: Panel/Report Entry Level subtopic, and the online help.



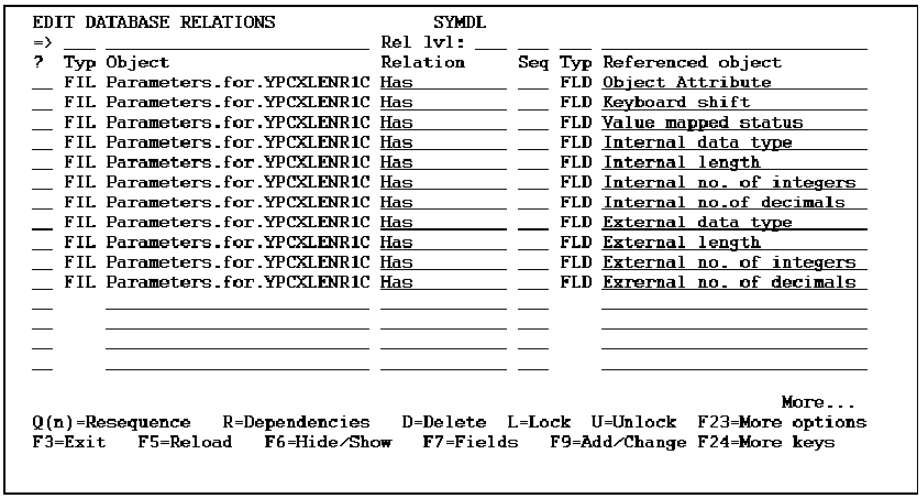
Example: Defining a Real Percentage Field (PCX)

Following is an example that illustrates the creation of the real percentage field type (PCX) with external field length calculation.

The example shows the steps required and all the panels involved in the order you would follow to create a user-defined field type in CA 2E.

The percentage data type has different internal and external field lengths, the external field length being determined from the length assigned to the internal version of the field. If the external field length is 5.2, the internal length becomes 5.4 to account for automatic division by 100. A percentage entered or displayed as 45.55, will be stored internally as 0.4555.

The external length calculation is performed by a program which derives its name from the data type. For a data type PCX, a program YPCXLENR1C must exist for this length calculation. Several parameters are required on this function. The figure below shows a generic STR type file in CA 2E, defining what fields are required as parameters to the length calculation program. This STR file should be defined on the Edit Database Relations panel as shown below. The file name contains XXX indicating the values to be substituted for the data type mnemonic.



```
EDIT DATABASE RELATIONS                    SYMDL
=> ___ ____________________________ Rel lvl: ___ ___ ___ ____________________________
?  Typ Object                      Relation      Seq Typ Referenced object
__ FIL Parameters.for.YPCXLENR1C Has           ___ FLD Object Attribute
__ FIL Parameters.for.YPCXLENR1C Has           ___ FLD Keyboard shift
__ FIL Parameters.for.YPCXLENR1C Has           ___ FLD Value mapped status
__ FIL Parameters.for.YPCXLENR1C Has           ___ FLD Internal data type
__ FIL Parameters.for.YPCXLENR1C Has           ___ FLD Internal length
__ FIL Parameters.for.YPCXLENR1C Has           ___ FLD Internal no. of integers
__ FIL Parameters.for.YPCXLENR1C Has           ___ FLD Internal no.of decimals
__ FIL Parameters.for.YPCXLENR1C Has           ___ FLD External data type
__ FIL Parameters.for.YPCXLENR1C Has           ___ FLD External length
__ FIL Parameters.for.YPCXLENR1C Has           ___ FLD External no. of integers
__ FIL Parameters.for.YPCXLENR1C Has           ___ FLD Exrernal no. of decimals
__     ____________________________ ____________ ___     ____________________________
__     ____________________________ ____________ ___     ____________________________
__     ____________________________ ____________ ___     ____________________________
__     ____________________________ ____________ ___     ____________________________

                                                                          More...
Q(n)=Resequence   R=Dependencies   D=Delete  L=Lock  U=Unlock  F23=More options
F3=Exit   F5=Reload   F6=Hide/Show   F7=Fields   F9=Add/Change F24=More keys
```

Once you have entered the relations, press F10 to define the file and fields as indicated in the figure that follows.

```
 EDIT FILE ENTRIES                        SYMDL

  File . . . . . : Parameters for YPCXLENR1C

? Field                     Type      Ocr  Et Ksq GEN name   Length    Renamed

_ Object Attribute          CDE           A      ASCD           3
_ Keyboard shift            CDE           A      ATCD           1
_ Value mapped status       CDE           A      AUCD           1
_ Internal data type        CDE           A      AVCD           1
_ Internal length           NBR           A      APNB         5.0
_ Internal no. of integers  NBR           A      AQNB         2.0
_ Internal no.of decimals   NBR           A      ARNB         1.0
_ External data type        CDE           A      AWCD           1
_ External length           NBR           A      ASNB         5.0
_ External no. of integers  NBR           A      ATNB         2.0
_ Exrernal no. of decimals  NBR           A      AUNB         1.0


SEL: Z-Details, R-Replace field, U-Usage, M-Mapped field parameters, L-Locks.
F3=Exit
```

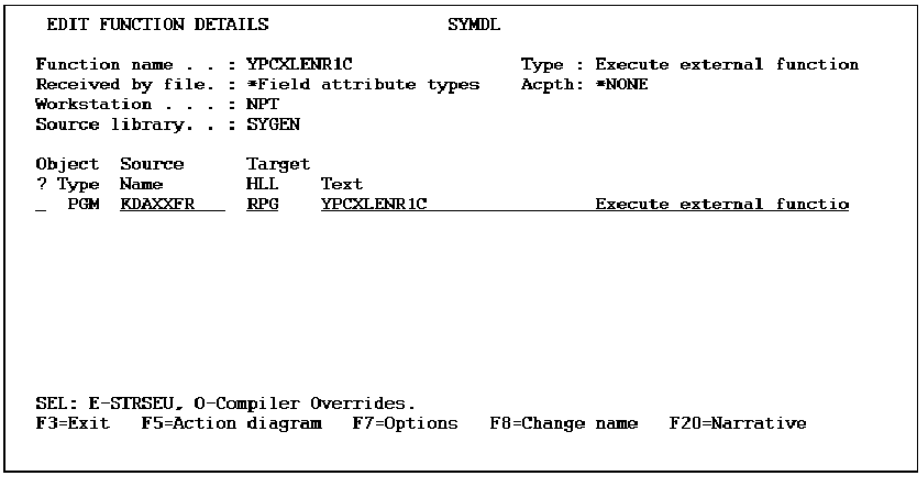
The external fields that are referenced fields are referenced to their internal counterparts.

The function to perform the external field length calculation can be attached to the *Field attribute types file. Position the Edit Database Relations panel at this file and go into the functions. Add one of the correct names, in this case YPCXLENR1C, making it of type EXCEXTFUN, as below.

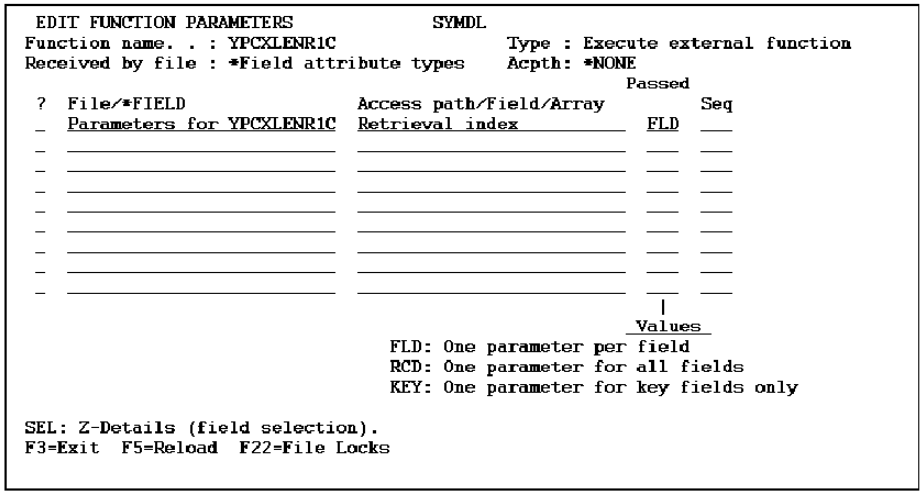
```
 EDIT FUNCTIONS                           SYMDL
File name. . . : *Field attribute types                  **   1ST LEVEL  **
 ____________________________ ____________________ ____________________________
? Function                   Function type        Access path
_ PCX Divide PCT by 100      Execute internal function *NONE
_ PCX Multiply PCT by 100    Execute internal function *NONE
_ YPCXLENR1C                 Execute external function *NONE
_ ____________________________ ____________________ ____________________________
_ ____________________________ ____________________ ____________________________
_ ____________________________ ____________________ ____________________________
_ ____________________________ ____________________ ____________________________
_ ____________________________ ____________________ ____________________________
_ ____________________________ ____________________ ____________________________
_ ____________________________ ____________________ ____________________________
_ ____________________________ ____________________ ____________________________
_ ____________________________ ____________________ ____________________________
_ ____________________________ ____________________ ____________________________
_ ____________________________ ____________________ ____________________________
                                                                        More...
SEL: Z-Details, P-Parms, F-Action diagram, S-Device design, N-Narr, O-Open,
  T-Structure, A-Access path, U-Usage, G/J-Generate, D-Delete, C-Copy, L-Lock.
F3=Exit        F5=Reload                  F7=File details   F9=Add function
F17=Services   F21=Copy *Template function
```

Zoom into the function and change the source member name to that of the desired length calculation program (YPCXLENR1C).

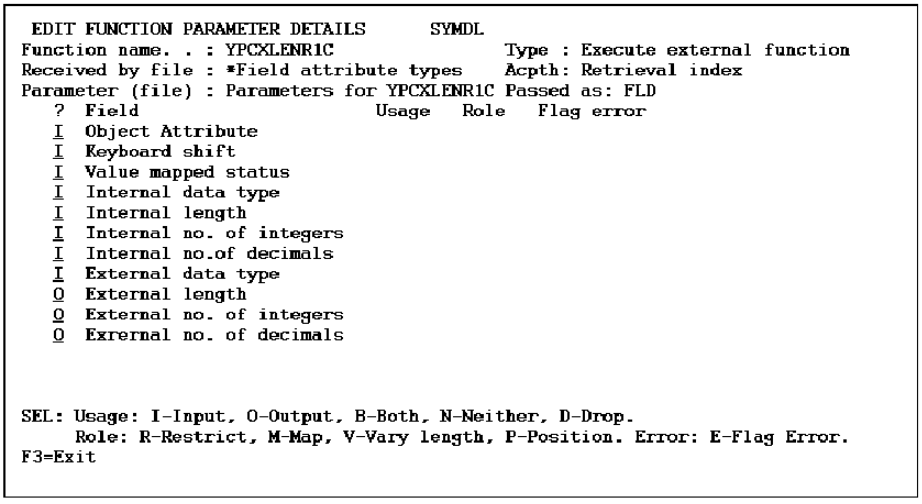
Attach parameters as defined by the structure file below, and give them the usage as indicated.



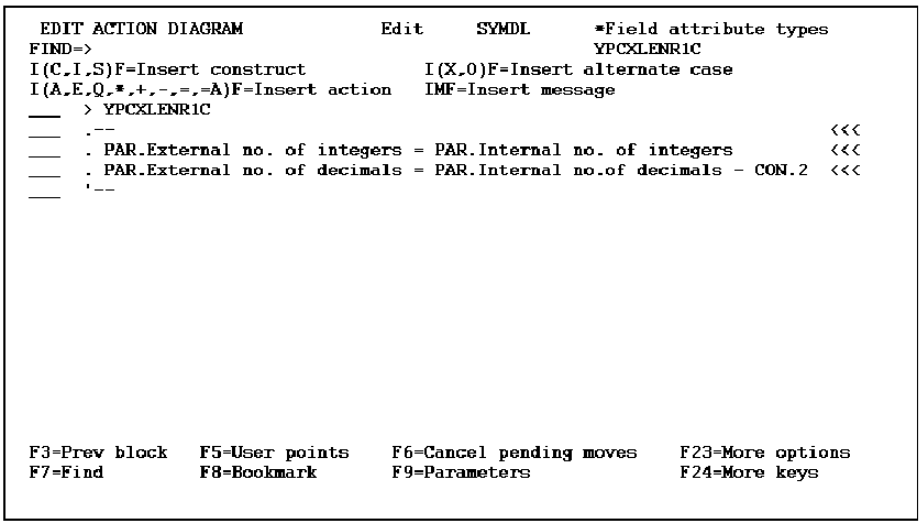
Attach parameters as defined by the structure file, shown below, and give the usage as indicated.



The parameters should all have a usage of I except the External length, External no. of integers and External no. of decimals. These are the parameters that the CA 2E calling function is expecting to be returned.

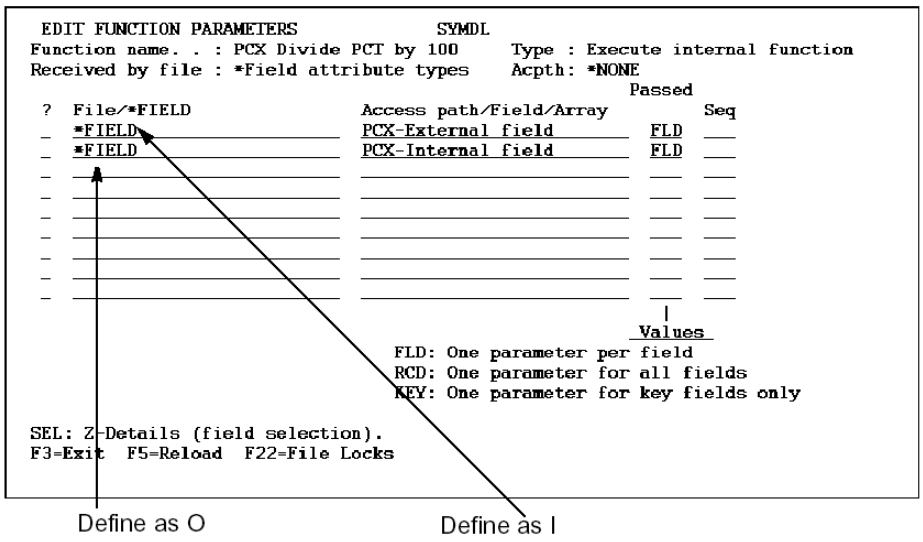


In this simple example, all the length conversion program will do is ensure that the number of decimal places externally is two less than the number internally. Set up the action diagram for the function as below. Then exit and save the function.

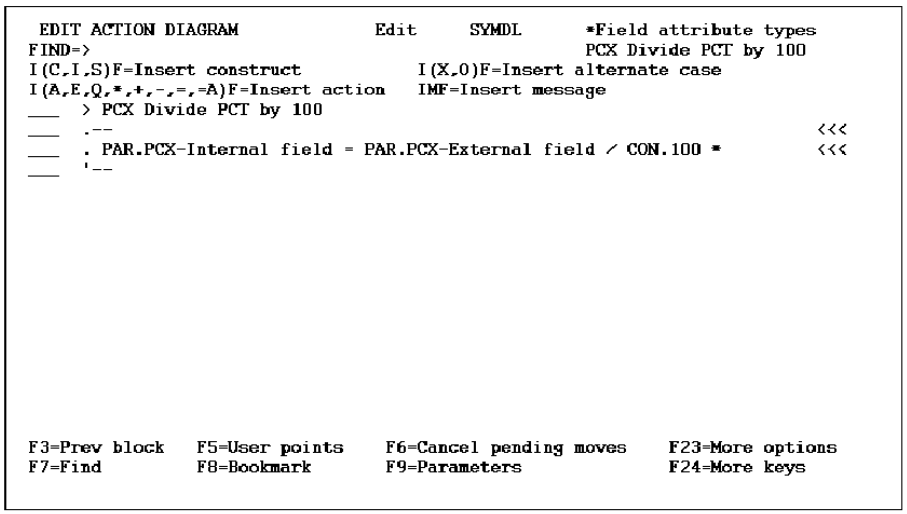


Create the Int/Ext and Ext/Int mapping functions now. These should be of type EXCINTFUN. In this case 'PCX Divide PCT by 100' and 'PCX Multiply PCT by 100' have been created for the mapping functions (see previous figure for Edit functions over *Field Attribute type file). Attach parameters to the functions, as indicated below. Defines both the PCX_External field and the PCX_Internal field as NBR type fields. Their lengths are arbitrary.CA 2E will create fields of the appropriate length at function generation time.

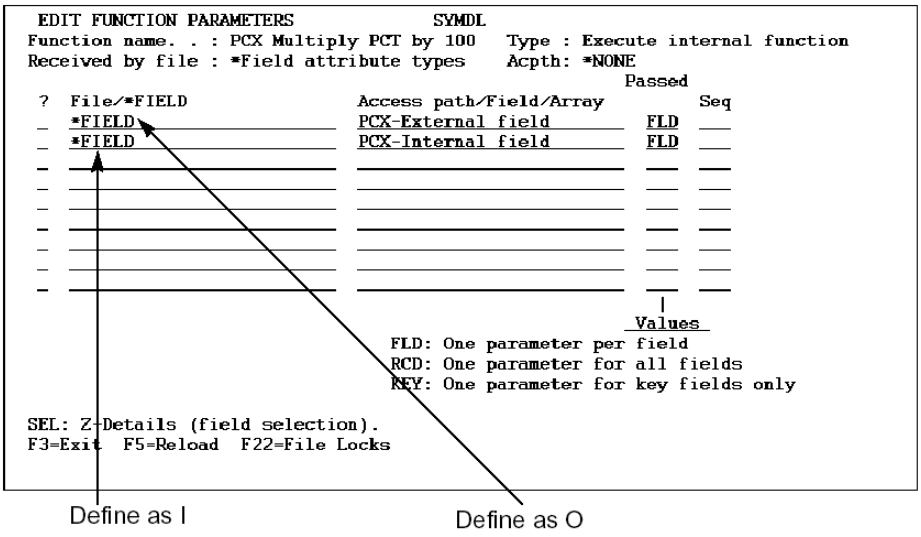
Ext/Int mapping function parameters:



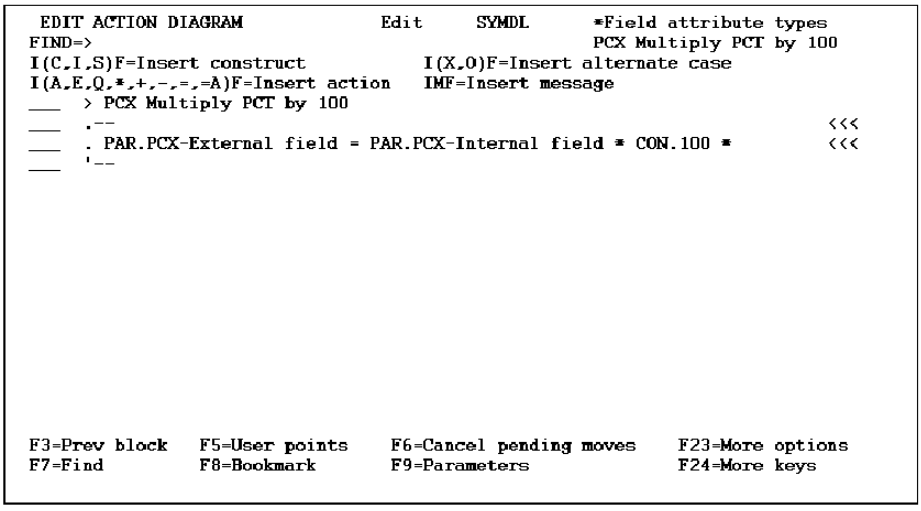
The Ext/Int mapping function should contain action diagram statements to divide the percentage by 100. Set the statement as below.



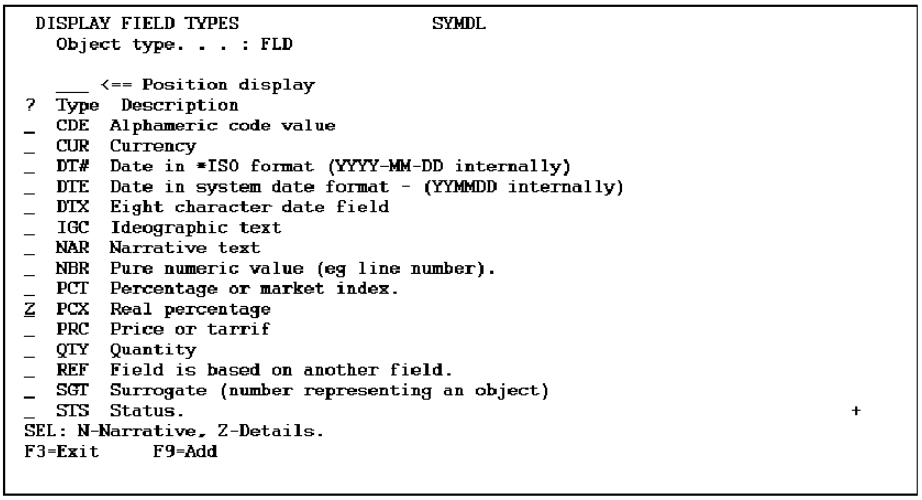
Int/Ext mapping function parameters:



The Int/Ext mapping function should contain action diagram statements to multiply the percentage by 100. Set the statement as below.

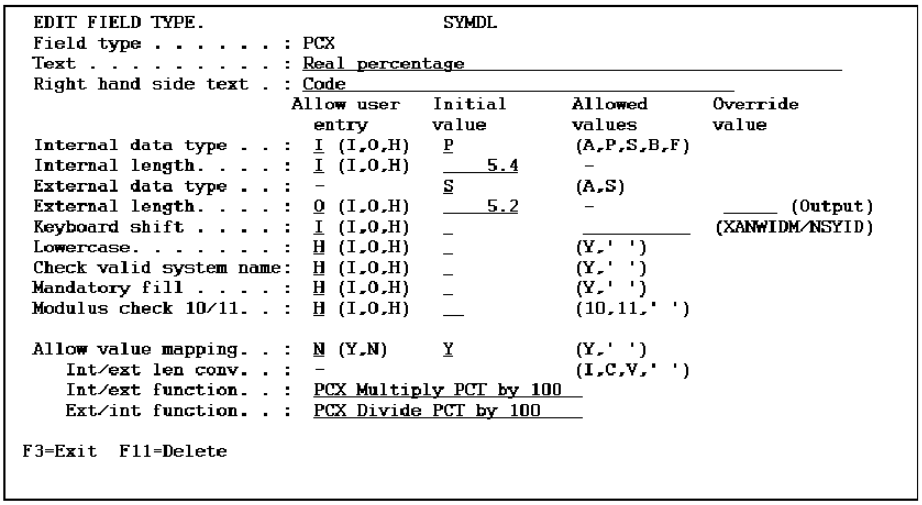


Finally, create the data type. Zoom into the *Field Attributes types file and press F9 to add an object attribute.



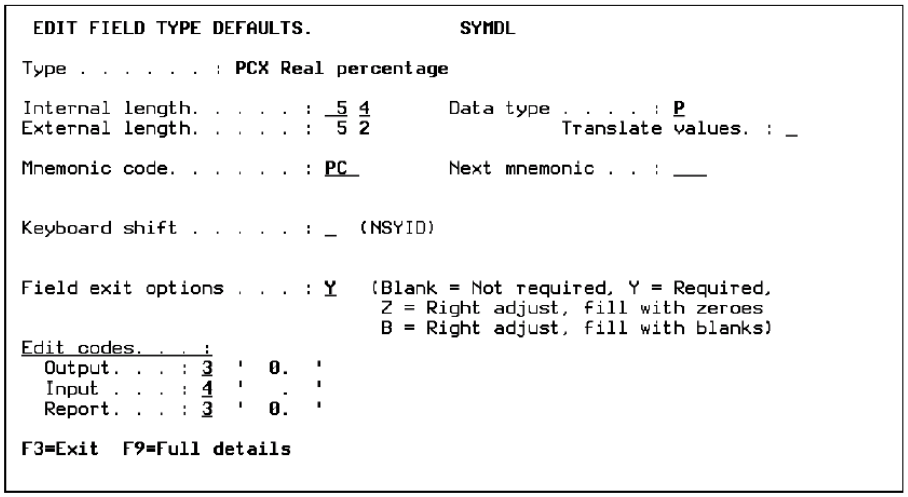
Enter Z against the PCX field type to display the Edit Field Type panel.

Create the data type as defined below, ensuring that the Int/Ext len conv field is set to **C** and that the correct mapping functions are specified.



Note: The YPCXLENR1C program must exist before you confirm this step; otherwise you will receive an error message.

Press Enter to display the summary panel. Change the Mnemonic code to PCX. Fill the field exit options and edit codes to complete the data type.



Chapter 6: Documenting Your Data Model

This chapter covers the CA 2E documentation commands you can use to produce hard copy documentation of your data model.

A document produced in this manner consists of details of the specific model's design objects that you want. The documentation provides a historical record of the development of your data model.

This section contains the following topics:

Related Information

This section covers only documentation of files, fields, relations, and application areas. For more information about documentation commands and how to use them to document access paths, functions, model library files, and messages, refer to the following modules:

- *Implementation Guide*
- *Command Reference*
- *Building Access Paths*
- *Building Applications*

Documenting Files, Fields, Relations, and Application Areas

Use the following procedures to document your model's files, fields, relations, and application areas. You must use the appropriate documentation command.

- Document Model Files (YDOCMDLF) command for files
- Document Model Fields (YDOCMDLFLD) command for fields
- Document Model Relations (YDOCMDLREL) command for relations
- Document Model Application Areas (YDOCMDLAPP) command for application areas

You can access these commands via the CA 2E Services Menu or by typing the command name directly from a command line on your screen.

CA 2E Documentation Commands

These are CA 2E documentation commands that you can use to document your data model's objects:

YDOCMDLREL	model relations
YDOCMDLF	model files
YDOCMDLFLD	model fields
YDOCMDLAPP	model application areas

All CA 2E documentation commands have several selection parameters and print options which allow you to:

- Select the type of information you want to be documented for the selected objects. For example, you can list files by a specific application area code, list the relations by the desired level, or print relation entries.
- Decide whether to include user-defined text in the documentation for each selected object. For example, you can use the PRTTEXT parameter to specify whether you wish narrative text to be included in the documentation listing, and if so, which type of text (functional or operational).

For more information:

- On narrative text, see the chapter "Using Your Model" in the *Administration Guide*
- On how to add narrative text at file, field, or relation level, see the chapter "Maintaining Your Data Model"

Using Documentation Commands via Display Services Menu

1. On the Edit Database Relations panel, press F17 to go to the Display Services Menu.
2. Select the Documentation menu option and press Enter.

   The Display Documentation Menu panel displays.
3. Select the appropriate option to document your model objects from this panel.

   Depending on the option you selected, CA 2E displays one of the following panels:

   - Document Model Files (YDOCMDLF)
   - Document Model Fields (YDOCMDLFLD)
   - Document Model Relations (YDOCMDLREL)
   - Document Application Areas (YDOCMDLAPP)
4. Change any of the default parameters as you wish and press Enter.

   A message displays indicating that the documentation is created.

   Note: If you press Enter without changing the defaults, CA 2E assumes that you want all of these options for your documentation.

   You will be returned to the Display Documentation Menu.

   To view the documentation online, see Viewing the Documentation procedure that follows.
5. Send your files to the printer according to your system configuration to obtain printouts.

Using Documentation Commands from a Command Line

1. Type YDOCMDLF (or YDOCMDLFLD, YDOCMDLREL, YDOCMDLAPP) and press F4. This will print the document using the default values.

   F4 shows parameter values.
2. Follow steps 3 through 4 of the procedure for using documentation commands via the Display Services Menu above.
3. From the Display Documentation Menu panel, type the option, and press F4 or Enter.

   Note: F4 allows you to change the default parameters on this panel. If you press Enter, CA 2E assumes that you want all of the default options for your documentation.

   A message displays indicating that the documentation is created.

   You will be returned to the Display Documentation Menu.

   Send your files to the printer according to your system configuration to obtain printouts.

Viewing the Documentation

1. From the Display Documentation Menu panel, press F8 to obtain a command line.
2. On the command line, enter the i OS Work with Job (WRKJOB) command. Select option 4 to display spool files created. Documentation is in spool file YDOCMDLxxx, where xxx identifies the type of report.
3. Type 5 next to the file you want to view and press Enter.

The following pages give examples of documentation listings.

Documentation Commands Output Listings

These are examples of the documentation produced from using the documentation commands.

YDOCMDLF (Docudel Fil*OCMDLFLD* (Document Model Fields)

COOL:2E air ticket model DOCUMENT FIELD DETAILS Op: MY WRKSTN1 05/08/92 20:39:57 Page: 1

Options: Model fields . . . . . *USER
Narrative text type *NONE

Model: Y2AIRMDL COOL:2E air ticket model

Field		Type Length	
Airline code		CDE 6 Field conditions : *NONE	DDS name: ABCD
	Referenced by:	Known by FIL Airline	
Airline name		TXT 25 Field conditions : *NONE	DDS name: ACTX:
	Referenced by:	Has FIL Airline	
Airport code		CDE 6 Field conditions : *NONE	DDS name: AACD
	Referenced by:	Known by FIL Airport	
Airline name		TXT 15 Field conditions : *NONE	DDS name: AATX
	Referenced by:	Has FIL Airport	
Arrival time		TME 6.0 Field conditions : *NONE	DDS name: ABTM
	Referenced by:	Has FIL Flight	
Date of issue		DTE 6.0 Field conditions : *NONE	DDS name: ABDT
	Referenced by:	Has 3 FIL Ticket	
Departure airport code		REF 6 Field conditions : *NONE	DDS name: AGCD
Departure airport name		REF 15 Field conditions : *NONE	DDS name: AITX

END OF REPORT

YDOCMDLREL (Document Model Relations)

COOL:2E air ticket model DOCUMENT RELATION DETAILS OP: MYWRKSTN11 05/08/92 Page: 1

Options: Model relations . . . *USER
Application area(s) *ALL
Narrative text type *NONE
Relation level. . . . *ALL
Print entries. . . . . *NONE

Model: Y2AIRMDL COOL:2E air ticket model

File/Item	Verb	Seq	File/Item		For	Sharing
FIL Airline	Defined as		FIL	Airline	REF	
FIL Airline	Known by		FLD K	Airline code	CDE	
FIL Airline	Has		FLD A	Airline name	TXT	
FIL Airline	Includes		FIL	Address	STR	
FIL Airport	Defined as		FIL	Airport	REF	
FIL Airport	Known by		FLD K	Airport code	CDE	
FIL Airport	Has		FLD A	Airport name	TXT	
FIL Airport	Includes		FIL	Address	STR	
FIL Flight	Defined as		FIL	Flight	REF	
FIL Flight	Owned by		FIL	Airline	REF	
FIL Flight	Known by		FLD K	Flight no.	CDE	
FIL Flight	Refers to		FIL	Airport	REF	Departure
FIL Flight	Refers to		FIL	Airport	REF	Destination
FIL Flight	Has		FLD A	Departure time	TME	
FIL Flight	Has		FLD A	Arrival time	TME	
FIL Flight departure	Defined as		FIL	Flight departure	REF	
FIL Flight departure	Owned by		FIL	Flight	REF	
FIL Flight departure	Known by		FLD K	Departure date	DTE	
FIL Flight departure	Refers to		FIL	Plane	REF	
FIL Flight price	Defined as		FIL	Flight price	REF	
FIL Flight price	Owned by		FIL	Flight	REF	
FIL Flight price	Known by		FLD K	Flight class	STS	
FIL Flight price	Qualified by		FLD K	Effective date	DTE	
FIL Flight price	Has		FLD A	Flight price	PRC	
FIL Ticket	Defined as		FIL	Ticket	REF	
FIL Ticket	Owned by	1	FIL	Airline	REF	
FIL Ticket	Known by	2	FLD K	Ticket no.	CDE	
FIL Ticket	Has	3	FLD A	Date of issue	DTE	
FIL Ticket	Refers to	4	FIL	Flight departure	REF	
FIL Ticket	Refers to	5	FIL	Flight price	REF	
FIL Ticket	Has	6	FLD A	Passenger name	NAR	

END OF REPORT

YDOCMDLAPP (Document Model Application Areas)

COOL:2E air ticket model Date: 05/08/92

DOCUMENT MODEL APPLICATIONS Page: 1

Options: Application area(s:) *ALL
Narrative text type : *NONE

Model: Y2AIRMDL COOL:2E air ticket model

Application Area:	SYS	COOL:2E air ticket model	
Files:		Address	STR
		Airline	REF
		Airport	REF
		Flight	REF
		Flight departure	REF
		Flight price	REF
		Plane	REF
		Ticket	REF
Application Area:	AIR	Airport and Airline	
Files:		Airline	REF
		Airport	REF
		Plane	REF
Application Area:	FLT	Flight	
Files:		Flight	REF
		Flight departure	REF
		Flight price	REF
Application Area:	TKT	Ticket	
		Ticket	REF

END OF REPORT

Chapter 7: Assimilation

This chapter describes assimilation and provides instructions for assimilating i OS database files using the Retrieve Physical Files into Model (YRTVPFMDL) command. It also lists the considerations and restrictions associated with this process.

This section contains the following topics:

Understanding Assimilation

Assimilation is the technique of retrieving and using file definitions from existing i OS physical files to create CA 2E files. The assimilated files then can be used in a CA 2E data model. You can add extra relations and define access paths and functions for assimilated files the same as you would with other modeled files.

Assimilating database files allows you to develop new functions and access paths over an existing database defined outside of CA 2E.

Degrees of Assimilation

Depending on your objectives, you can choose to carry out assimilation to these different degrees:

- Assimilation as is

  The existing database is preserved exactly as is. Field, format and file names remain the same. You will be able to use the files with all your existing programs without a need for change or recompilation.

  This degree of assimilation, however, may lead to restrictions in certain cases, particularly if the data model implied by the existing database is incorrect or inconsistent.

- Assimilation with limited modifications

  The existing database is preserved but model data structures can be rationalized or corrected where appropriate. You will still be able to use most of your existing programs with only minor modifications.

  You can take full advantage of CA 2E capabilities now because your files have been fully defined in the model and can be used as any other modeled files. The greater the degree of assimilation that can be achieved, the greater the use that can be made of CA 2E capabilities when generating programs.

For more information on normalization and considerations regarding normalized databases, see the chapter "Developing a Conceptual Model."

Using the YRTVPFMDL Command

You use the Retrieve Physical Files into Model (YRTVPFMDL) command to perform assimilation. Confirm that all developers and programmers are out of the model. Execute the YRTVPFMDL command before entering the model.

The YRTVPFMDL command retrieves into a CA 2E design model all file and field definitions not already present in the model. It creates a CA 2E file for each i OS physical file retrieved and a CA 2E field for each of the retrieved fields found with a unique DDS name. The fields are connected to the file with Has relations.

The YRTVPFMDL command has several parameters that allow you to perform functions such as removing the DDS prefix from field names, amending a retrieved field, and nominating a file that you want CA 2E to treat as a field reference file.

Note: You can retrieve an i OS SQL table into your model by specifying the name of the associated i OS physical file in the library for the SQL collection.

Parameters/Functions

Parameters for the YRTVPFMDL command are as follows:

- FILE specifies qualified generic name of the i OS physical file from which the description is to be retrieved.
- RMVFLDPFX specifies whether a prefix will be removed from the implementation field names in the retrieved file.
- REFFILE specifies the name of the file to be treated as the field reference file. No Has relations will be created for the file. All fields will be added to the field dictionary.

For more information refer to the YRTVPFMDL command, the CA 2E *Command Reference*.

Adding Extra Information to Assimilated Files

A file created through assimilation may not contain as much information as one of your normally modeled files. You may have to add extra information to each of the assimilated files. For example, because the YRTVPFMDL command does not identify the key fields of a file that it assimilates, you may have to identify the key fields of that file to CA 2E by changing some of its relations from Has to Known by. Be aware that if the relations are not sequenced, the Known by relations are placed before the Has relations. This changes the file format, making it different than the existing physical file. To keep the same field sequence, you must resequence the physical file format entries as detailed in the next topic.

Editing i OS Physical File Format Entries

CA 2E stores default file format entries for fields on an assimilated file, including:

- sequence of the field
- implementation name of the field on the physical file or the SQL column name on the table
- i OS data type of the field
- length of the field

You can view and change all of these defaults using the Edit Physical File Format Entry panel, which is available only for assimilated files. To access this panel:

1. Zoom into the assimilated file from the Edit Database Relations panel. The Edit File Details panel displays.
2. Zoom into the PHY access path of the file. The Edit Access Path Details displays.
3. Zoom into the access path format entries. The Edit Access Path Format Entries panel displays.
4. Press F8 to get the physical file entries. The Edit Physical File Format Entry panel displays.

This panel enables you to enter and maintain details about the fields that belong to the format of an assimilated physical file. It shows the name and type of format. If adding a relation causes a mismatch between the sequence of fields defined to CA 2E and the existing physical file, change the sequence number defined on the Edit Physical File Format Entry panel. For example, if a Known by relation causes the field to be defined as one of the first fields in the file, change the sequence number defined on the Edit Physical File Format Entry panel.

Considerations

There are some considerations associated with assimilation. Resulting from the way CA 2E implements a data model, considerations involve

- i OS field names and field naming in CA 2E
- Inconsistent implicit data model
- Date formats

Whether these considerations will prove restrictive depends on

- The extent to which you wish to preserve the existing database
- Names used in the existing database
- The extent of the "correctness" of the model as implied by the assimilated set of i OS files

In making this implied model explicit, assimilation tends to highlight any inconsistencies such as missing keys, redundancy, conflicting field lengths, or the association of an attribute with the wrong entity.

Changing Field Name and Attribute Type

Prefix

If the names of your fields in each file already contain a prefix, you can tell CA 2E to remove the prefix by specifying a value of *YES for the RMVFLDPFX parameter on the YRTVPFMDL command.

Duplicate Field Names

Existing i OS physical files may contain the same field name in several different files. For example, you may find a field named Order Date that could well mean Customer, Purchase, or Shop Order Date. If you have this problem, you should either:

- Use the REF field attribute to base the definition of one of the fields on that of the other, or
- Use the Edit Physical File Format Entry panel to establish a different field name for the field on the file.

Inconsistent Implicit Data Model

If the data model implied by your existing database is incomplete or inconsistent, you may want to rationalize and correct it before you can take advantage of using 's CA 2E's capabilities.

Examples of Inconsistency

Examples of inconsistency are:

- **Different field definitions.** If you have used different attributes for a same field in different files, then you must standardize the field definitions before you can establish a CA 2E relationship. For instance, if Customer code is defined as five characters in one file and six in another, then you must make the length the same for both instances of this field.
- **Missing field entries.** You may have omitted a field from a file that is necessary to resolve a relation, either by mistake or because the omitted field is to be supplied procedurally.

  In particular, you may have omitted key fields if you have used either relative record processing or purely arrival sequence processing, neither of which are supported by CA 2E data modeling.

Date Formats

If you wish to use the CA 2E date field type (DTE), you may need to convert the date field's data and length to match the internal stored format described by CA 2E. Dates of this field type are always stored on file in CYYMMDD format. CA 2E provides automatic date validation and mapping into the local display format.

CA 2E assimilates i OS DATE fields as field type DT#.

For more information on date field types, see the chapter "Understanding Your Data Model" and *IBM IBM i Programming: Data Description Specifications Reference*.

Using Extended by Relations in Assimilated Files

You can use the Extended by relation to add more data to an assimilated file without actually changing the file. The Extended by relation allows you to specify a one-to-one relationship between the existing file and a new, extended file that contains the additional fields you want to add. You will get automatic validation and existence checking and be able to define virtual fields from either file into the other.

Using Extended by relations also helps you save disk space because you can choose only to create records in the extended file when and if the data is needed.

Example of Using Extended by Relations

You have assimilated your Customer master file and do not want to change it, but you want to add some credit history data for those customers who finance their purchases.

In your data model, add the new relations as shown below:

Customer	**Extended by**	Customer Credit History
Customer Credit History	**Owned by**	Customer Master
Customer Credit History	**Has**	Avg Payment Days
Customer Credit History	**Has**	Avg Payment Amount
Customer Credit History	**Has**	Last Payment Date

The Extended by relation does not generate new fields in the Customer file so you do not have to change your programs. You have to maintain only the Customer Credit History records associated with the Customer file.

Assimilation Procedure

Use the following procedure to perform assimilation.

1. **Retrieve existing i OS physical files** using the YRTVPFMDL command. This command allows you to retrieve (the definitions of) one, several, or all i OS physical files in the database.

You can specify a value for the RMVFLDPFX parameter of the command to indicate whether or not you want CA 2E to drop the prefix from the DDS field names in the retrieved file. The purpose of using this parameter is to make sure the DDS names match those of your existing programs.

For more information, see the section Changing Field Name and Attribute Type.

2. **Identify key fields.** You identify key fields to CA 2E by changing the relevant Has relations into Known by relations since all fields are assimilated as Has relations.

3. **Adding file-to-file relationships.** You change or add new relations to explicitly indicate file relationships to include the processing logic for the keys arising from Owned by or Refers to relations, and for referential integrity checking.

If adding a relation causes a mismatch between the sequence of the fields you defined to CA 2E and those of the existing i OS physical file, you can change the sequence number using the Edit Physical File Format Entry panel. For example, if a Known by relation causes the field to be defined as one of the first fields in the file, change the sequence number using the Edit Physical File Format Entry panel.

4. **Lock the physical file.** You lock the physical file in your model to prevent inadvertently selecting the existing file definitions for recompilation.

For more information on:

- Physical files, see the Building Access Paths chapters "Adding Access Paths" and "Tailoring For Performance"
- How to lock files, see the chapter "Using Your Model" in the Administration Guide
-
-
-

Index

E

F

P

Q

R

S

T

U

V

W

Y