Visual LANSA Logical Modeler

- About this Guide
- What's New in this version?
- Introduction
- Getting Started
- Models
- Datatypes
- Entities
- Elements
- Relationships
- Abstractions
- User Views
- Building a Model
- Troubleshooting
- Tutorials

Edition Date October 20, 2014 © LANSA

About this Guide

- This guide provides instructions for planning and creating a data model using the LANSA Logical Modeler.
- The contents are written for system analysts and LANSA developers and assumes that readers have a solid understanding of data modeling and are familiar with one or more data modeling methodologies.
- How to install the Visual LANSA and Microsoft Visio software (which is used for the display of the model) is beyond the scope of this guide. It is assumed that all the prerequisite software is loaded and configured as described in the *Installing LANSA on Windows Guide*.
- Although there is a chapter summarizing data modeling principles, this guide is not designed to teach you how to model data. If you are unfamiliar with data modeling, you should attend the *LANSA Modeling and Design Workshop*.

You can access the Visual LANSA guides from the Help menu on the Visual LANSA menu bar.

Before using this guide, you should review the following:

About this Version Who Should Use This Guide

How To Use This Guide

Additional Information.

For the latest product information and updates, please refer to the technical information section of the LANSA Web site at www.LANSA.com/support

Note: The information in this guide is specific to the current LANSA Logical Modeler.

About this Version

Logical Modeler requires a currently supported version of Visio. Refer to the Supported Platforms and Versions document for the currently supported version of this software.

Models produced by previous versions of the Logical Modeler are fully supported by this version.

A summary of features in this version:

Models Entities Elements Relationships Abstractions Model Navigation Miscellaneous

Models

- In addition to an XML file being generated, a Visio document is created. This enables models produced by this version to be loaded more quickly than in previous versions.
- A right-click menu is available on a blank portion of the model diagram for performing common model tasks.

Entities

- Physical file names for data and variant entities may be up to eight or ten characters in length, depending on your Visual LANSA system settings.
- If an entity's physical file name is longer than two characters less than the maximum number of characters allowed by the system settings, the entity's relationships and user views must have logical view names specified manually.
- If an entity's physical file name is longer than two characters less than the maximum number of characters allowed by the system settings, logical view names for the entity's relationships and user views are optional. If no logical view name is specified, the modeler will automatically generate a logical view name consisting of the entity's physical file name and a sequential two digit number, as with previous versions.
- You can perform a one-time import of an existing physical or logical file (an External Entity) definition. External entities may be related to logical entities in order to extend existing databases.
- Entity validation is performed dynamically, indicating whether or not the entity is in error prior to build processing.
- The way in which an entity is shown as having a validation or build errors is shown as an inverted red triangle above the top, right-hand corner of the entity shape.

Elements

- Field names for elements may be up to nine characters in length.
- Notes may be maintained for any element inherited by an entity as the result of a relationship. As a result, notes may be contextualized according to the intended use of the inherited element.

Relationships

- Element field names inherited by entities via relationships are edited using the Maintain Entity Elements form.
- A Logical View Name may be specified for appropriate relationships. It is optional if the entity's physical file name to which it belongs is up to six characters in length. It must be specified if the entity's physical file name is more than six characters in length. If it is not specified, the modeler will automatically create a logical view name, based on the entity's physical file name and a sequential two digit number.
- Relationships may be specified by using the "Specify Relationship..." option on the Edit pull-down menu or main toolbar.
- Relationships may be deleted via the Maintain Entity Elements dialog.
- You can choose whether or not relationships are displayed on the model diagram.

Abstractions

- The position of entities in an active abstraction may be changed independently of the main model diagram.
- Objects may be added to and removed from an active Abstraction.

Model Navigation

- A "Go To" facility simplifies the search for an entity in a large model.
- A grid may be toggled on and off to make it easier to align model shapes.
- How you nominate the view to be shown for the model and/or individual entities is simple.
- Entities and relationships can be created by means other than dragging shapes from the document stencil onto the model diagram.

Miscellaneous

- Some Builder Queue terminology has changed.
- The nine most recently opened models may be opened directly from the File pull-down menu.
- The modeler uses standard Windows terminology, and dialogs for opening and saving models.
- Model shapes have Windows-style right-click menus.
- Context-sensitive user guide help is provided from all Logical Modeler dialogs.

Who Should Use This Guide

This guide has been written for system analysts and developers who are responsible for designing or redesigning applications.

It is assumed that the reader has some knowledge of Visual LANSA, LANSA development and the LANSA Repository.

Visual LANSA and the Logical Modeler can be used to model and build databases on many different platforms including IBM i, Windows and Linux.

A brief summary of the techniques of data modeling has been included, however, this guide is not intended as a detailed explanation and some knowledge of data modeling or data analysis is assumed.

The Logical Modeler uses extended Chen E-R modeling principles. It is important that you are familiar with data modeling terms and concepts before attempting to use the Logical Modeler.

For a detailed explanation of Data Analysis or Modeling, it is recommended that you refer to one of the many books on the subject, or attend the LANSA Modeling and Design Workshop.

How To Use This Guide

This guide contains detailed instructions for the use of the LANSA Logical Modeler.

- For a general overview and notes to review before you start using the modeler, refer to the Introduction in order to become familiar with LANSA's methodology and terminology.
- A tour of the Logical Modeler's main window is provided in Getting Started.
- The various facilities you will use when creating your model are described within the heading to which they relate. For example, for all Element details, refer to Elements, for Relationship information, refer to Relationships and so on.
- A complete set of Tutorials are provided with the Logical Modeler. These tutorials provide an excellent overview of the features of the Logical Modeler.
- A sample model is shipped with the modeler. The sample model is named **LANSA Logical Modeler Example Model.ldm** and is stored in the LANSA directory where the LANSA software is installed. In order to load the sample model correctly, Visual LANSA should use English as the development language. In addition, no computer names have been specified for any of the elements or entities in the model. This ensures that none of your existing repository definitions can be overwritten by the building of the model as shipped.
- If you encounter any problems when using the Logical Modeler, refer to Troubleshooting.
- To access online help for the Logical Modeler, use the Help Menu from the main window or the F1 key to display the context sensitive help.

Additional Information

For more details about the Visual LANSA software, you may need to refer to:

- *Installing LANSA on Windows Guide* supplied with the LANSA online guides on the LANSA Windows CD and installed with the Visual LANSA software.
- *Visual LANSA User Guide* supplied with the Visual LANSA software.

If you are unfamiliar with data modeling, you should attend the LANSA Modeling and Design Workshop. For more details, contact your local LANSA distributor or check the LANSA Web site.

For the latest product information, refer to the LANSA product Web site at www.LANSA.com/support

1. Introduction to Data Modeling

The LANSA Logical Modeler allows a system analyst to design and manipulate a complete logical data model using elements, entities and relationships and then allows a physical database to be built from this model.

This section defines what data modeling is, in relation to the LANSA Logical Modeling tool. It also describes how to normalize data prior to creating your model and your physical database files.

Even if you are familiar with data modeling, please review this chapter briefly:

- 1.1 What is Data Modeling?
- 1.2 Conceptual versus Implementation Modeling
- 1.3 How Does LANSA Help with Data Modeling?
- 1.4 Conventions/Terminology
- 1.5 Overview of the LANSA Logical Modeler

If you are an experienced data modeler, it is strongly recommended that you review the 1.4 Conventions/Terminology to help you understand how to use the LANSA Logical Modeler.

1.1 What is Data Modeling?

Data Modeling is a technique used in the Data Analysis phase of the Software Development Life Cycle. It is used by system analysts or designers to build a picture (or model) of the data used within an existing or proposed system, based on information gathered from users. The normal data modeling process involves the analyst gathering information from users about the data used in their normal business activities (i.e. their system).

This information is presented in the form of "structural statements" or "business rules". It enables the data structure to be described in concise terms which the users can easily understand. The users are then able to review and verify the data model. For example, consider a company which manufactures a variety of different engineering products (screws, bolts, etc.). Some of the business rules which apply to their customer order processing system might be as follows:

- A customer may place many orders over a period of time.
- Each order may include many products.
- For each product on the order, a quantity is specified.
- A customer has a name, an address and a phone number.
- Each product is stored in a particular location within the warehouse.
- Orders are numbered sequentially for each customer.
- A product is known by its product code.
- A customer is identified by a unique customer number.

Each of these statements can be interpreted as describing the associations between two or more items, for example, Customer and Order. Some of the descriptions in the business rules simply state that elements belong to entities (as attributes). For example, a customer has a name, an address and a telephone number. Other statements describe how entities can be identified. For example, a product is known by its product code. A third group of statements describes the relationships between entities. For example, a customer may place many orders.

The items described will fall into two categories. Those items which are basic data items and cannot be subdivided (for example, Telephone Number) are known as elements. The other type of item is actually a group of data elements and is known as an entity (for example, Customer or Product).

The initial stages of data modeling identify the data elements, to group them into entities, and to show how each occurrence of an entity can be uniquely

identified. It is very important to ensure that data elements are correctly grouped into entities and the techniques used for this purpose are generally known as Data 1.1.12 Normalization.

Once entities and elements have been defined within the model, relationships between entities are introduced. A relationship shows how to use data, given an occurrence or record from one entity, to access one or more occurrences in a second entity. For example, for a specific order, how it is possible to identify the customer who placed that order.

Review the following:

- 1.1.1 Proper Data Modeling
- 1.1.2 Logical Models versus Physical Models
- 1.1.3 Understanding Entities, Attributes and Elements
- 1.1.4 Understanding Attributes and Relationships When Modeling
- 1.1.5 Understanding Relationships
- 1.1.6 Results of Relationships
- 1.1.7 Identifying the Type of Relationship
- 1.1.8 Understanding Identifying Attributes
- 1.1.9 Relationship Guidelines
- 1.1.10 Understanding Shared Keys
- 1.1.11 Approaches to Data Modeling
- 1.1.12 Normalization
- 1.1.13 Training for Data Modeling

1.1.1 Proper Data Modeling

Most developers spend their time maintaining existing applications rather than building new applications. Consequently, they are accustomed to adding to existing databases rather than developing new ones. When an opportunity arises to develop a new application, the approach is quite often implementation oriented.

Many developers are familiar with designing and building physical databases rather than working with conceptual models. They tend to create a storage facility to support their programs. It is a direct step from data to physical database model or database.

This approach may appear effective, but can result in a number of problems:

- poor application performance
- higher application maintenance costs
- database is not easily understood by the users or developers
- database is not flexible
- database is not normalized
- applications cannot be easily integrated.

A database must be properly designed before it is implemented. A database is best implemented using a logical or conceptual data model that starts with the application business requirements.

Note the following definitions:

- A logical data model is a representation of a physical database.
- A database is an implementation of a data model.

A logical data model overcomed some of the problems that occur when developing a database directly from the data.

Some developers claim to use data models and modeling tools but they are actually just implementing a physical model or database design. Conceptual modeling involves more than using graphical drawing tools to create a picture of the database. Models start with business requirements and relationships within the business. Models begin with the abstract and then build toward the physical.

1.1.2 Logical Models versus Physical Models

A model is a representation or a simplified copy of something more complex. Architects use models to examine large or complicated structures which are difficult to work with directly. For example, engineers test models of air plane wings or create scale models of buildings.

A data model is a tool which allows the Architect to construct a picture of the data within an organization or a part of an organization. It is not a picture of what the computer sees, but of what the user and business sees.

Logical Data Models are high level or conceptual models which describe data in terms of the user perspective and business rules.

A logical data model represents the business.

A logical data model looks at the basic pieces of information and the interdependencies between the information. Logical data models are defined in terms of entities, attributes, and relationships. They reflect the business, its information and rules. It is not designed to represent the Architect's perception of the physical database to be implemented.

A logical data model is platform independent. It is not based on hardware, software, or tool. Logical data models are a description of the business information and not a physical database design.

Physical data models or database designs are low level models which describe data in terms of files and how the information is stored in the computer. When physical modeling, you think of header and detail files, logical views of data, joins of data, etc.. Physical models reflect the technical implementation of the database.

A physical data model represents the computer database.

Physical models look at data files and database design in terms of their implementation. Physical models are defined in terms of fields, record formats, and files. It is a technical representation designed to support the programs and programmer that will use the database.

A physical data model is usually platform dependent. It is designed for a specific computer platform. The hardware, database management system, software language and tool used are all factors in developing a physical model.

As a result, physical models tend to be very inflexible. A physical model cannot be easily moved from one environment to another, or implemented on another hardware platform.

1.1.3 Understanding Entities, Attributes and Elements

The Entity-Relationship (E-R) Model was developed by Chen in 1976. It attempts to structure data in a way which can be easily understood by the user. Some of the basic terms include:

Element: An element is the smallest piece of data within a system. An element cannot be divided into other pieces of information. Example: Age is an element. Name is not an element, as it can be broken into first name and last name.

Attribute: An element which is associated with (attached to) an entity becomes an attribute of that entity. Example: An age can be an attribute of entity Employee. An Employee entity has an attribute of last name. An Employee entity has an attribute of first name.

Entity: An entity is a group of elements which logically belongs together. Example: An Employee is an entity. An Order is an entity.

Value: A value might be a number or set of characters an attribute or element may be assigned. Example: The age of an Employee is 25 years. The Employees surname is 'Smith'.

Occurrence: A single example or instance of an entity (i.e. a row in a table) is called an occurrence.

Identifying or Key Attribute: A key attribute is one whose value uniquely identifies an occurrence within an entity. Example: An employee number identifies an Employee. An order number identifies an Order.

To summarize:

- Age is an element.
- Age becomes an attribute of the entity Employee.
- The entity Employee is made up of attributes such as employee ID number, first name, last name, and age.
- A specific Employee is identified by the employee ID number.
- The age of an employee may have a value of 25 years.

An occurrence within the Employee entity is:

- Employee number has value 123.
- Employee first name is 'John'.
- Employee last name is 'Smith'.
- Employee age is 25 years.

1.1.4 Understanding Attributes and Relationships When Modeling

The following is the most important rule to remember when creating a logical data model:

An attribute may be associated with, or attached to, only one entity in your data model.

If the same attribute is required in another entity, it must be the result of a relationship.

An attribute such as customer number may be associated with the Customer entity. Once the customer number attribute is attached to the Customer entity, it cannot be attached to any other entity in your data model. The customer number cannot be made an attribute of the Order or Invoice entities. If the Order entity requires a customer number to identify the customer which placed the order, than a relationship must be created between Order and Customer.

The Attribute Rule is very important for two reasons:

- The Attribute Rule will ensure that your data model is properly normalized.
- The Attribute Rule will ensure that your data model reflects the structure of your data and the business.

The Attribute Rule helps to create a logical data model instead of a physical model of files in the database. An attribute such as customer number cannot be added to more than one entity without using relationships to define the structure of the data.

It is important to note the difference between assigning the same attribute to more than one entity and having many similar attributes as part of many entities. You must decide if an attribute is actually related data from another entity or a separate piece of data.

1.1.5 Understanding Relationships

The relationship is an important concept within logical models. Relationships are used to define the structure of your data. Relationships connect one entity to another. They show how information from one entity is used to access information in another entity.

A relationship can be defined as follows:

A relationship exists between two entities if it is possible to use data from the first entity to access data from the second entity.

For example, a Customer entity contains all the information about a customer. The Order entity contains all the information about an order. An Order is placed by a Customer. There must be a link between the information in the Customer entity and the information in the Order entity. A relationship is used to establish this link.

In basic Chen E-R models, there are two types of relationships: Parent/Child Relationship and Join/From Relationship. Each relationship involves a different set of business rules and produces a different structure in terms of the final database.

Parent/Child Relationship

This type of relationship exists if the data in one entity is owned by another entity or one entity is contained within another entity. For example:

Order is an entity. Order Line is an entity.

Relationship: An Order may have one or more Order Lines. The Order Lines belong to a specific Order.

Join/From Relationship

This type of relationship exists if the data in one entity is used to access or "look up" data in another entity. For example:

Customer is an entity. Order is an entity.

Relationship: An Order refers to a Customer. The Customer places an Order.

The relationships not only define the structure of the data within the model, they actually affect the data or attributes within the model. The use of relationships, in combination with the attribute rule, ensure that our data is properly structured.

1.1.6 Results of Relationships

Relationships affect the attributes of the entities which are involved in the relationship. When specifying a relationship between two entities, all of the key attributes of one entity may be transferred to the other entity. The transferred key attributes are referred to as either parent keys or join attributes. The following definitions apply:

Parent Key

A parent key is an attribute(s) which is present in a child entity as a result of a parent/child relationship.

Example: An Order Line entity has a parent key of order number, i.e. order number is an attribute of Order Line because of a parent/child relationship with the Order entity.

Join Attribute

A join attribute is an attribute(s) which an entity contains as a result of a join relationship.

Example: An Order entity has a join attribute of customer number from the Customer entity.

The importance of the parent keys and join attributes is made clear when you consider the attribute rule.

1.1.7 Identifying the Type of Relationship

How do you know if a parent/child or join/from relationship should be used?

Parent/Child

One entity is said to be the Parent of another if the data of the Child entity belongs to (is contained within) the parent. Parent/child relationships can be identified by asking "Is the entity identifier unique?" or "Can the entity exist on its own?"

If the relating entity's identifying attribute is unique, i.e. you can identify an occurrence in the entity solely by its identifying attribute, then the entity probably has no parents. If you require other information to identify an occurrence, i.e. another entity must exist before it can exist, then the entity probably has a parent.

For example, an Order Line entity is identified by its Line Number. Line Number is not sufficient to uniquely identify an occurrence within the entity. There can be several Order Lines with a Line Number of 001. You need an Order Number and the Line Number to identify a specific occurrence in Order Line. Order Line is a child of Order.

Also note that the parent entity, Order, is required or the Order Line cannot exist. You cannot have an Order Line without an Order.

Join/From

One entity is said to be joined (or to refer) to another if a data (i.e. non key) attribute or group of attributes from the first entity can be used to access an occurrence of the second. Again ask, "Is the entity identifier unique?" or "Can the entity exist on its own?"

The join relationship differs from the parent relationship, in that the attribute(s) used to "look up" the referenced entity is not necessary to identify the primary (or referencing) entity.

For example, in the order processing example, an Order Line refers to a Product. You would expect the Order Line to contain some reference to Product such as a product number. However, the Order Line Number and the Order Number are sufficient to uniquely identify the Order Line. The reference to Product is not required to identify an occurrence in Order Line. The relationship is simply a join.

Also note that a Product Code does not have to exist in order to have an Order Line. A valid value may be required for the product code (i.e. you refer to the

other file), but the Order Line Entity could exist without having a Product entity. A parent 'gives birth' or contains a child. The Order contains the Order Lines. The Product does not contain the Order Lines.

1.1.8 Understanding Identifying Attributes

Here are some other important points to remember when working with identifying attributes of an object.

Identifying Attribute Rules:

- An entity does not have to have an identifying attribute. In some instances, an entity may be identified solely by its parents or may not require any identification.
- An entity may have only one identifying or key attribute. It is important to note that more attributes may be required to uniquely identify an occurrence within the entity, i.e. you require the parent keys along with the identifying attribute.
- If an entity has more than one parent, it does not require an identifying attribute. The two parent keys will identify the child entity. For example, a Customer is the parent of Customer Products and Product is the parent of Customer Products. Customer Products is an entity listing which customers have which products.
- Parent keys are compounded. A parent key may be made up of more than one attribute. For example, a Company is a parent of a Department which is a parent of Employee. Employee has two parent keys: Company Code and Department Code.

1.1.9 Relationship Guidelines

Rules for relationships are often imposed by modeling tools. Following are some general relationship guidelines:

- An entity cannot have a parent relationship with itself, nor can it have as a parent, any entity to which it is directly or indirectly a parent.
- An entity cannot have an entity as a parent if it already has one of that entity's children as a parent. Such a relationship would be superfluous. It can, however, join to such an entity.
- An entity may have more than one parent relationship with another entity.
- An entity may have a join relationship with any other entity, including itself.
- An entity may have more than one join relationship with another entity.
- An entity may have both a parent and join relationship with another entity.

Note: The relationship rules or guidelines enforced will depend upon the data modeling tool used.

1.1.10 Understanding Shared Keys

Another important consideration when modeling is the sharing of key attributes. When entities are involved in multiple relationships, the key attribute fields may be repeated in an entity. It is important to know whether the attribute keys are repeated or shared.

Consider the following Bank example.

- Bank is an entity.
 - Bank is known by its Bank Code.
 - Bank has a name.
 - Etc...
- Bank is the parent of Account.
- Account is an entity.
 - Account is known by its parent's Bank Code.
 - Account is known by Account Number.
 - Account has an amount.
 - Account has a name.
 - Etc...

Now imagine that you need to track transactions involving a specific Bank and Account. One transaction is called a Transfer which involves moving money from one Account to another.

So if keys are not shared, the model will be:

- Transfer is an entity.
- Transfer refers to a debit Account.
- Transfer refers to a credit Account.
- Transfer is an entity.
 - Transfer is known by its Transfer ID Number.
 - Transfer has a join attribute Debit Bank Code.
 - Transfer has a join attribute Debit Account Number.
 - Transfer has a join attribute Credit Bank Code.
 - Transfer has a join attribute Credit Account Number.

- Transfer has an amount.
- Transfer has a date.
- etc.

In this situation, transfers can be made between two different Banks (or within the same Bank). The Debit Bank Code and Credit Bank Code can be different. The parent key of Bank Code is NOT shared. If the business rules specify that transfers cannot be made between banks, then the Bank Code must be the same for each account. The parent key of Bank Code should be shared. If the parent key is shared, then the model will be:

- Transfer is an entity.
 - Transfer is known by its Transfer ID Number.
 - Transfer has a join attribute Bank Code.
 - Transfer has a join attribute Debit Account Number.
 - Transfer has a join attribute Credit Account Number.
 - Transfer has an amount.
 - Transfer has a date.
 - Etc.

Note that two relationships with the same entity can never share all their keys. In our example, this means that Transfer could not share both Bank Code and Account Number in both its relationships. This would have no meaning.

1.1.11 Approaches to Data Modeling

Two approaches are commonly used to build data models:

- top down and
- bottom up.

The top down approach begins with identifying the entities within the business. Entities are the major items which the users identify or work with. For each entity identified, create a list of attributes which make up that entity. Review the attributes to be sure that they cannot be broken into smaller pieces of data, i.e. they may be entities themselves. In this way, start at the highest level and fill in the details below.

The bottom up approach begins with identifying all the elements in the system. This is a very long and difficult process of attempting to identify all the pieces of information the user comes in contact with. Once the elements are identified, they are grouped together to create entities.

The top down approach is recommended because it more closely relates to how the users define their business. Following the top down approach may also prevent you from falling into the trap of physical modeling rather than logical modeling.

For example, a user may tell you that an order lists the product to be shipped to a customer. There is no mention of an order line number. An Architect might introduce the concept of an order line number simply because a key is needed for a database file. This is physical modeling.

1.1.12 Normalization

In 1972, Ted Codd from IBM developed the relational model and identified normal forms which were called first, second, and third. When normalizing a database, you are ensuring that it has been broken into its normal forms. Normalization is a common technique for analyzing relational data structures. It is important to understand the basic concept of normalization when data modeling. Normalization applies to physical database models.

The benefits of having a properly normalized data model and database design are that it:

- Reduces data redundancy.
- Produces a database structure that is easily understood by users and system developers.
- Produces a data model that is flexible in the face of changing business requirements.
- Produces a database structure that encourages structured, and therefore simpler, program development.

The process of normalization involves three main steps. These are:

First Normal Form	Remove repeating groups. Remove from an entity those elements which maintain the same value between occurrences (or records) while the rest of the elements change.
	Remove elements which are only partially dependent on the key of the entity.
Third Normal Form	Remove elements which are dependent on (i.e. are identified by) a key other than that of the entity.

The result of the normalization process should be that each entity contains only those elements which are properly identified by the keys of that entity.

For example, consider the information which might appear on an order form. It could start with an entity which contains all the data items such as in this example:

Un-normalized Form

First Normal Form

First remove Repeating Groups. The customer details will be the same for each product ordered (i.e. for each line of the order). They are separated from the ORDER information which will be different on each line.

ORDER	Order Number, Customer Number, Customer Name, Customer Address	The fields in bold are the unique or candidate key
ORDERED PRODUCT	, , ,	fields.

Second Normal Form

Next remove attributes only partially dependent on primary key (or part of compound key). The product name depends only on Product Number (not on Order Number) and so it is removed.

ORDER	Order Number, Customer Number, Customer Name, Customer Address	The fields in bold are the unique or candidate key
ORDERED PRODUCT	Order Number, Product Number, Quantity Ordered	fields.
PRODUCT	Product Number, Product Name	

Third Normal Form

Finally, remove attributes dependent on a key other than the primary (or

compound key). The customer details depend only on the customer number and so are removed.

ORDER	Order Number, Customer Number	The fields in bold are the
ORDERED PRODUCT	Order Number, Product Number, Quantity Ordered	unique or candidate key fields.
PRODUCT	Product Number, Product Name	
CUSTOMER	Customer Number, Customer Name, Customer Address	

1.1.13 Training for Data Modeling

This guide is not designed to teach the principles of data modeling.

If you are unfamiliar with data modeling, you should attend the LANSA Modeling and Design Workshop.

The LANSA Modeling and Design Workshop includes the following topics:

- Application Development
- Data Analysis
- Data Models
- More Data Modeling
- Extended Chen E-R Models
- C-Think, I- Think
- Implementing the Data Model

The LANSA Modeling and Design Workshop provides an excellent introduction to logical modeling concepts and will help you to properly use logical models when building databases for your business.

Please contact your LANSA agent for more details.

1.2 Conceptual versus Implementation Modeling

In the following section, the concepts of C-Think (conceptual) and I-Think (implementation) are explained. These concepts describe the approach which System Architects may use when modeling application systems.

Review the following:

- 1.2.1 Thinking About Modeling
- 1.2.2 Understanding C-Thinkers
- 1.2.3 Understanding I-Thinkers
- 1.2.4 Dual C/I-Thinkers
- 1.2.5 Thinking about Dates
- 1.2.6 C-Think Considerations
- 1.2.7 I-Think Considerations
- 1.2.8 New World C-Thinkers

1.2.1 Thinking About Modeling

Knowing the benefits of using logical or conceptual data models, how should organizations use logical models to achieve optimal results? The most significant influence on properly using logical models is the "approach" and "attitude" towards modeling.

Approach refers to how you work with and create your model. Do you model a business or do you simply build a database? Are you considering the users' needs or the programmers' needs. Attitude is reflected by your objectives and how you use the model. Do you want to understand the business and design an application with a future, or do you want a database that you can justify right away?

For one organization, an optimal result is an implemented database that can most easily grow and adapt to meet the changing business needs. For another organization, an optimal result is an implemented database that delivers the best possible execution speed, no matter what the cost is in the future. For most organizations, the optimal result lies somewhere between these extremes.

A C-Thinker will use a model to document the business and its rules. C-Thinkers use words like "ENTITY", "ATTRIBUTE", and "RELATIONSHIP". Their model reflects the business and can be used by the users to verify, test, and alter the organization of data. When working, they are said to be using C-Think techniques.

An I-Thinker will use a model to document a database design that they have already planned. I-Thinkers use words like "FILE", "HEADER", "DETAIL", "FIELD", and "RECORD". Their model is implementation oriented and designed for programmers. When working, they are said to be using I-Think techniques.

Most developers begin as I-Thinkers. By understanding the nature and role of the C-Thinker, and by using C-Think techniques, they begin to see things differently. During the process of evolution, an I-Thinker will become a Dual C/I-Thinker. Eventually the C-Thinker role should grow and develop until it takes over and the Architect becomes a New World C-Thinker.

1.2.2 Understanding C-Thinkers

C-Think refers to Conceptual thinking. C-Thinkers do not focus on the complex uses for physical files, logical files, indexes, join files, spanned logical files, multi-member files, select/omit criteria, dynamic selection options, open query files, etc. C-Thinkers attempt to produce a model that truly represents the information that is used by the business not the computer. It is to be a "model" of what happens in the "real" or "fuzzy" world, not a platform dependent database implementation.

The C-Thinker regards the modeler as a note pad, or drawing board. The model aids in the recording, analyzing and validating their ever changing ideas. Sometimes purist C-Thinkers produce optimal results. This most likely occurs in an organization with a very high "performance budget", that values accuracy, quality, design portability and the ability to evolve with rapid business changes above all else.When a purist C-Thinker converts their model to an implementable database it will usually be:

- Built from "end user" objects/approaches concepts (OOD)
- Completely normalized
- Easy to understand (for other developers)
- Well liked and understood by the end users
- Able to adapt to change very easily
- Easy to generate applications that use it
- Easy to maintain applications that use it
- Near to or outside of the organization's "performance budget".

1.2.3 Understanding I-Thinkers

I-Think refers to Implementation thinking. I-Thinkers focus on how to make their previous database design adapt to a business requirements change by using 4 join files with 3 concatenated and 5 sub-stringed partial join keys. I-Thinkers often create complex database structures that are difficult to maintain. I-Thinkers generally design physical databases. They work from their own experiences, sometimes extended by the grandfather-father-son learning technique.

The I-Thinker treats the modeling with caution, because it is really just a tool for entering their data descriptions to build a file. (This is a design approach actively encouraged by some modeling tools.) Sometimes purist I-Thinkers produce optimal results. This most likely occurs in an organization with a very low "performance budget", that values low computer resource usage above all else.

Most purist I-Thinkers implement a database that:

- Is built from DP concepts/approaches
- May be only partially normalized
- May be hard to understand (for other developers)
- The end users may or may not like the applications that use it
- May not adapt to change very easily
- May not be easy to generate applications that use it
- May not be easy to maintain applications that use it
- Will run well inside an organization's "performance budget"

1.2.4 Dual C/I-Thinkers

Dual C/I-Thinkers combine both mind sets when modeling. They have not learned to free themselves of I-Think, and yet they understand C-Think. In the Dual C/I-Think stage, the most important things to understand are:

- When to work as a C-Thinker
- When to work as an I-Thinker
- How to put aside I-Think considerations and reservations in the search for the "perfect" conceptual model
- How to take a C-Think database design and alter its implementation so that it will work within a performance budget and its conceptual integrity remains intact.

Once the Dual C/I-Thinkers understand the when and how, they will quite quickly evolve into a New World C-Thinker when modeling and designing applications.

1.2.5 Thinking about Dates

In C-Think mode, you would define a date using three separate entities: Year, Month and Day. Year in the parent of Month. Month is the parent of Day. Hence the Day is identified by parent keys from Year, Month and its own identifying attribute for Day (YY MM DD).

Now several relationships may exist between an entity called Order and the entity called a Day. For instance, an Order has a "Date Order Placed" and "Delivery Date". However, what often happens is that the I-Think part of the Architect's brain takes over and consequently:

- The elements "Date Order Placed" and "Delivery Date" are placed directly into the entity Order.
- The user understands what a date is in elemental form, but decides that they do not want to have three database files defined to represent the entities Year, Month and Day.

The idea of the model is that it conceptually represents what information is being used within the model. If a user chooses not to have database files representing the entities Year, Month and Day, the decision should be made at the time the logical model is being implemented as a database (I-Think time). Making the decision in advance (during the modeling) may lead to problems.

Consider the 3 entity C-Think form of the model. What will happen once a Century entity is defined? Century is the parent of Year. Now the Day entity is identified by CC YY MM DD attributes. The change will ripple through the whole model and Date Order Placed and Delivery Date will use the CCYYMMDD form, but not if they were added directly as elements to the Order entity.

This is one of the most powerful features of data modeling - to make a major change to the structure of the model and have the effects automatically "cascaded" or "rippled" through all other affected relationships.

So, an I-Thinker will critically examine the entity Order and find that it includes the following attributes:

- Century Order Placed Delivery Century
- Year Order Placed Delivery Year
- Month Order PlacedDelivery Month
- Day Order PlacedDelivery Day

This may horrify the I-Thinker, who will think:

- There's going to be 8 separate fields here. I don't want all those extra fields in the database.
- Those 8 fields are going to increase the effort in my order processing programs. I'll be forever sub-stringing them together and apart again so that they come up as one field on screen panels and reports.

What the Architect should think (using C-Think) is that:

- This is a good model. It represents how the data is structured and the rules for the data.
- As an I-Thinker, I'm not too crazy about all these data fields, but I'll cross that bridge when I come to it.

When they come to the implementation phase, they will probably find that other facilities exist to very easily and consistently map the dates into one virtual field for processing on screens and reports. In the long term, they will probably find that the choice to implement the 8 separate fields in the resulting database pays off.

1.2.6 C-Think Considerations

The Classic Order Model

By using C-Think, you can often look at the way things are done (or have always been done) in a completely new way. Sometimes this method confirms the correctness of your existing approach.

For example, consider a classical order entry model. An Order entity is the parent of an Order Line entity where the Order Line entity is identified by a parent Order Number and Order Line Number attributes. This implementation can probably be made to work. However, ask yourself:

- Is it correct?
- What business rules does it represent?
- What business rules does it enforce?
- Where did the element "Line Number" come from?
- Who invented it?
- What is it used for?

Using C-Think, we may produce a different model for the order model. Perhaps the Order and Product entities are the parents of Ordered Product. An Order Line is not really required. In this type of model, the Ordered Product entity does not have an identifier of its own as it can be identified by its two parent keys, Order Number and Product Number. There is no need for an Order Line Number.

Product Code Example

In many databases, you will encounter an attribute called a "Product Number". This attribute is often the identifying attribute of the an entity called Product. However, at some time later, you hear mention of Product Groups and Product Classes that are part of this Product Number identifying attribute. Ask yourself:

- Where are they?
- What are they?
- Where do they come from?

You may soon discovered that these groups and classes are actually the embedded digits within the Product Number. (Consider how telephone numbers are really combinations of different groups of digits that have different meanings.) A C-Thinker would never leave Product entity identified by such a Product Number. The Product Group and Product Class become entities with relationships to the Product entity so that the identifying attribute to Product has the proper parent keys.

1.2.7 I-Think Considerations

Dates

When you consider implementing the conceptual model for a date as four different entities (Century, Year, Month, Day), your implementation thinking will likely consider concatenating this information into a single date element. In many circumstances, a single date field is sufficient as the conceptual structure of the date is not required.

In I-Think mode, you should consider the "performance budget" of your organization. You must also consider the integrity of the conceptual model. Try to ensure that physical implementation changes do not affect the fundamental integrity of the conceptual model.

You may find that as you make I-Think changes, the C-Think part of your brain actually starts making comments such as:

- If I did implement this entity then it would be easier to ...
- I've never thought of it like that before ...
- If we implemented ... we could easily handle ... as well.

If this starts to happen, you have become a real dual role C-Thinker and are on the way to evolving into a New World C-Thinker.

Some classical I-Think implementation changes include:

- Omitting conceptual entities from the implemented database
- Concatenation of elements for a user field.

These types of changes can be made for performance reasons or implementation reasons based on the software language used.

Tables

Some additional I-Think implementation changes might include:

- implementing array structures
- implementing several entities into one physical table.

Consider the sales history of a company. Historical information is based on the product sold and the period in which it is sold. Rather than implement the database with 12 records for each month's product sales in one year, an array structure can be substituted. Now a product has one record for each year with 12 sales months in each record.

In a model you may also have a very large number of entities which are

basically used as "constants" or tables for validating data. Data such as city, state, country, company, or warehouse might be grouped together as a single table when implementing the database.

These types of changes should be based on a sound understanding of the current and future needs of the business and not solely on technical implementation.

I-Thinking for Existing Models

When working with an existing application database which does not have a logical model, you may choose to use an I-Think approach to create a model which represents the physical database. This approach is used assuming that you are not going to change the database structure of the existing application, but you want to build from it.

Modeling the physical database will quickly provide a definition of the entities and attributes you will need for creating your new model. Because you will not be altering the existing database, there is little benefit to develop a conceptual model which does not reflect the already implemented database. Your C-Think focus will be on the new entities, attributes, and relationships for the new applications you are designing.

1.2.8 New World C-Thinkers

To become a New World C-Thinker, you must:

- Be a C-Thinker whenever you are modeling
- Be an I-Thinker when implementing a model as a database.

The New World C-Thinker uses a modeling tool to build a purely conceptual model which reflects the business. When it comes time to implement the model, the New World C-Thinker uses I-Think techniques to make the implementation a success.

Many people are, by background, I-Thinkers. It takes some practice to understand and become a C-Thinker. When the C-Think part of your brain starts to argue with the I-Think part, you are starting to make the best use of a data modeling tool.

By working with two modes, starting with a C-Think mode and then later with an I-Think mode, you should be able to produce an optimal result for your organization at all times. When I-Thinker mode forces you to make an implementation change, at least you will be doing it from an informed point of view.

Over time, as your skills improve and computer hardware becomes cheaper and faster, you should expect and allow for the C-Think part of your brain to evolve and grow until it represents the only thought method you use. Pure I-Thinkers and pure C-Thinkers don't really exist, but there is some of them in all of us.

The examples and scenarios presented here have been simple, and serve to illustrate several points:

- Real conceptual modeling has something to offer us all.
- To use conceptual modeling most effectively, a different "mindset" is required.
- The only real way to achieve the required mindset is by a process of evolution. It cannot be obtained by revolution. The user must believe in the conceptual approach.
- A person working with the required mindset can make informed decisions about physical implementation concessions.

More and more, you should be acting in a C-Thinker role. I-Think should evolve to a dual I-Thinker/C-Thinker, and should finally evolve to become a New World C-Thinker.

1.3 How Does LANSA Help with Data Modeling?

A methodology helps you formalize your actions, whereas an application development tool helps you automate your actions. LANSA is methodology or approach independent. This simply means that regardless of the approach you plan to use to design your application, LANSA will assist you in building the application.

If you use the software engineering approach, you can use the Logical Modeler to define your database and then use LANSA's process, function and component structures and the rich development facilities to build your application. Using the various Frameworks provided with LANSA, you can rapidly build prototypes over the physical database created by the Logical Modeler.

If you use the information engineering methodology, you have LANSA's Logical Modeler to design and implement your information model and supporting physical database.

LANSA allows the system's analyst to build up and manipulate a complete logical data model comprising elements, entities and relationships and applies data analysis rules to ensure that the model is properly normalized. It then allows the physical database to be generated from the logical model, but recognizes the fact that there will be differences between the two. For example, for performance reasons, you may not wish to implement a fully normalized database.

LANSA documents the logical data model in such a way that it is understood by users. It can be reviewed and verified by the users as part of the analysis/design process before any physical database is built.

The LANSA Logical Modeler improves the developers productivity by automating the build of the physical database. The modeler helps the developer to create quality databases with a consistent standard. It supports the concept of a corporate data types and allows the developer to define standards for the implemented physical database.

1.4 Conventions/Terminology

There are a number of different methodologies for structured Analysis/Design and Data Modeling. Each has its own terminology. Most methodologies use the same underlying constructs, although they call them by different names. LANSA has selected the most useful constructs which are common to most methodologies.

The following list describes how some of the terminology is used in the LANSA Logical Modeler:

Abstraction	A container object used to group entities. It is not an entity. It is not used in relationships.
Attribute	An element which is associated with (attached to) an entity. For example, the Customer Name is an attribute of the Customer entity.
Cascade	If the identifying element (key) of an entity changes, then the keys of any relationships defined from that entity will change. For example, when the key of a Parent entity changes, the keys of any child entities will also change.
Element	A basic data item (sometimes referred to as a column in a table). If an element is not part of an entity, it is called an unattached element.
Entity	A group of elements which belong logically together and can be uniquely identified by a key or group of keys (also known as a table).
Identifying Element	The element of an entity which uniquely identifies each occurrence of the entity (either on its own or in conjunction with one or more parent keys). For example, Customer Number uniquely identifies each occurrence in the Customer entity. Also

called an Identifying Attribute.

Join An attribute which is used to refer to a joined entity, i.e. as a Attribute key in a join relationship (also known as a foreign key). For example, the Customer Number can be a join attribute in the Order entity.

Join A relationship between two entities such that one or more Relationship attributes or keys from one entity can be used to access the second entity. For example, the Order entity has a Join relationship to lookup data in the Customer entity.

- Occurrence A single example or instance of an entity (i.e. a row in a table or a record on a file). For example, "Acme Retail Company of Australia" might be an occurrence of Customer.
- ParentAn entity which owns another entity (the child), i.e. to whichEntitythe child entity belongs. For example, the Order and Product
entities are parents of Ordered Product entity.
- Parent Key An identifying element which is used as a key on the parent entity and therefore also on the child entity. For example, Ordered Product entity is a child to the parent Order entity. The key of Order (Order Number) is also a key of Ordered Product.

ParentA relationship between two entities such that the data in oneRelationshipentity is owned by another entity. For example, the Order entity
has a parent relationship to the Ordered Product entity.

Relationship A relationship exists between two entities if, given an occurrence of one entity, it is possible to use data from that entity to access one or more occurrences of the second entity. For example, given an order, it is possible to obtain information

about the customer who placed it.

User View A view of the data in an entity. User Views are used to define the keys used to create logical views over the implemented database file.

1.5 Overview of the LANSA Logical Modeler

In order to the use the LANSA Logical Modeler, you should be familiar with the following:

- 1.5.1 Models
- 1.5.2 Entities
- 1.5.3 Elements
- 1.5.4 Relationships
- 1.5.5 Abstractions
- 1.5.6 User Views
- 1.5.7 Building a Physical Database

1.5.1 Models

The Logical Modeler enables you to create data models based on the Chen E-R methodology. As such, a data model contains entities, elements and relationships. These three types of information combine to create a picture of the data that your business uses.

Once described, a data model can be built into a physical database. The physical definitions are stored in the LANSA Repository. The data model can be redesigned and refined over many build cycles, until a physical database is generated that matches the complete business requirements.

Data Models are stored as XML files and Visio documents, independently of the LANSA Repository. The models can be easily copied, moved or shared with other users.

For more details, refer to 3. Models.

1.5.2 Entities

Entities can be anything that a user works with in their business. For example, in an order entry application, a Product, a Customer or an Order may be entities.

Entities contain elements that describe their characteristics. For example, a Customer entity may have elements such as Customer Code and Customer Name.

Each entity is assigned a type. This is very important in determining how entities interact with each other via relationships, and how they are implemented in the repository when the model is built. The four types of entity are:

- Data Entity
- Conceptual Entity
- Variant Entity
- External Entity

For more details, refer to 5. Entities.

Data Entity

A Data Entity stores information, and is the most commonly used type of entity. For example, a Customer is a data entity.

Data entities are implemented as database tables in the LANSA Repository.

Conceptual Entity

A Conceptual Entity describes information but does not store information. For example, Address information might be modeled as a Conceptual Entity.

Conceptual entities are implemented as fields in the LANSA Repository.

Variant Entity

A Variant Entity is similar to a Data Entity in that it stores information. It is used in situations where elements are dependant on the value of another element. For example, there may be a type of Product called a Manufactured Product. Associated with a Manufactured Product may be information such as Drawing Number. There may be another type of Product called Supplied Product, which contains the Vendor who supplied the product.

Variant Entities are implemented as database tables in the LANSA Repository.

External Entity

An external entity is used to define a physical or logical file which already

exists in the LANSA repository. The key structure of this definition can be used in relationships with logical entities in order to extend the existing database. No database tables are implemented from External Entity definitions.

1.5.3 Elements

Elements are used to define data. Each element is a unique piece of data in the model. This data may become an attribute of any type of entity.

For example, a Customer entity might have attributes of Customer Name, Customer Code, Credit Limit etc.

Elements can be inherited by entities from other entities via relationships. For example, the Order entity is related to the Customer entity. Identifying elements from the Customer entity are inherited by the Order entity so that the Customer who has placed a particular Order can be identified.

Elements are implemented as fields in the LANSA Repository.

1.5.4 Relationships

Relationships are used to relate two entities to each other. One entity in a relationship will inherit elements from the other. The type of relationship will dictate the nature of this inheritance and how the entities are implemented in the LANSA Repository.

Relationships influence the file fields, file keys, logical files, referential integrity validation rules and access routes that are generated when an entity is built.

The different types of relationship available in the Logical Modeler are:

- Parent/Child
- Join
- Include
- Variation

For more details, refer to 7. Relationships.

Parent/Child

One entity is said to be the Parent of another if the data of the Child entity belongs to (is contained within) the Parent.

For example, an Ordered Product cannot exist on its own. The Ordered Product must have a parent, Order, before it can exist.

The identifying elements of the Parent entity are inherited by the Child entity. These elements are often called Parent Keys or Foreign Keys. They become the file's primary key when the entity is physically implemented in the LANSA Repository.

Join

One entity is said to be joined (or to refer) to another if a data (that is, non-key) element or group of elements from the first entity can be used to access an occurrence in the second entity.

The Join relationship differs from the Parent/Child relationship in that the element(s) used to access the referenced entity are not necessary to identify the primary (or referencing) entity. However, they are still inherited to become Foreign Keys.

For example, an Order is placed by a Customer. The Order is identified by its Order Number. Order will look up a Customer Code in the Customer entity. Hence, the Order file will contain the Customer Code element when it is physically implemented in the LANSA Repository.

Include

The Include relationship allows elements from one entity to exist in many different entities. However, unlike a Parent/Child and Join relationships, no foreign keys result. The elements become part of the entity.

For example, a conceptual entity called Address is created with attributes of Street, State, City, Country and Post Code. This entity has no identifying element. The Address entity could be included into a Customer entity or Supplier entity. The Address elements will become part of the Customer and Supplier files when they are physically implemented in the LANSA Repository.

Variation

This type of relationship can be used between a Data or External entity and a Variant entity where the Data or External entity is the source or the relationship and the Variant entity is the target.

As such, the Target entity inherits the Source entity's identifying elements. These elements become the Target entity's primary key when it is physically implemented in the LANSA Repository. The Variant entity can not have an identify element of its own.

1.5.5 Abstractions

An abstraction is like a container object and is used to a group entities. It is very useful when working with large data models. For example, an abstraction called Order Processing can be used to group the Customer, Product, Order and Ordered Product entities. By activating an abstraction, you can now work with just the Order Processing part of the model (four entities) rather than all entities in the model. Entities which are not part of the abstraction are not displayed in the diagram.

Once defined, an abstraction can be activated in order to simplify the model diagram view. In addition, it allows you to build only the objects contained within the abstraction, rather then the entire model.

For more details, refer to 8. Abstractions.

1.5.6 User Views

User Views are not a logical modeling concept. They are included to allow analysts to document logical views of the data which can be used by the physical database.

The aim of a user view is to allow you to define access to a file in a sequence other than the primary key and logical views created as a result of relationships. User views will generate logical views when a database build is performed on an entity.

For example, a logical view sequenced by Customer Name will not ordinarily be created for the Customer entity, as there are no relationships involving Customer Name and it is not part of the key of Customer. However, an end user may wish to view Customer information in Customer Name sequence. A user view can be created to support this requirement.

For more details, refer to 9. User Views.

1.5.7 Building a Physical Database

Once created, a model can be built into a physical database. The build process takes all of the element, entity and relationship information and creates fields and tables in the LANSA Repository. In addition, it creates logical views, access routes and validation rules as the result of relationships defined within the model.

Building a model consists of three phases for each entity.

Building Elements

Entity elements are implemented as fields in the LANSA Repository. They also become fields in the files that are generated from entity definitions.

Building Entities

Data and Variant entities are implemented as database tables in the LANSA Repository. Conceptual entities are implemented as fields only.

Entities that are implemented as tables will contain fields that have been created from the element definitions. In addition, fields may be created and attached to the file for elements that are inherited as the result of relationships with other entities.

Note that External entities are not implemented in the LANSA Repository, as they represent files that already exist.

Building Relationships

As well as relationships influencing the fields that are created for a database table, they are also implemented as referential integrity validation rules, logical views and access routes in the LANSA Repository.

For more details, refer to 10. Building a Model.

2. Getting Started

This section defines the software pre-requisites to use the Logical Modeler and how access the software. It also describes the various components of the main modeler window and how you interact with them when maintaining a data model.

Note: To access online help for the Logical Modeler, use the Help Menu from 2.4 The Main Window (the F1 function key will display the Visio help text).

2.1 Pre-Requisites

Visual LANSA Requirements

The Logical Modeler is integrated into the Visual LANSA development environment. It is executed at the partition level and can be used with single language or multilingual partitions.

It is recommended that the partition in which you execute the modeler contains the STD_ system fields. These are used by the modeler 4. Data Types when building the model. If the partition does not contain the fields which are referred to by the Data Types, errors will be generated when you attempt to build the database.

In order to use the Logical Modeler you will also need the following software:

Microsoft Visio

The Logical Modeler utilizes the Microsoft Visio software for its diagramming capabilities. Before you can use the Logical Modeler, Visio 2003 or a later version of Visio should be installed.

Microsoft's XML Parser Software for XML Compatibility

The Logical Modeler uses XML to store models and requires Microsoft's XML Parser software. The XML Parser software is installed with Visual LANSA and can be downloaded from the Microsoft Web site.

Internet Explorer

A currently supported version of Internet Explorer is required to use the Logical Modeler. Refer to the Supported Platforms and Versions document for the currently supported version of this software.

2.2 Models and LANSA Partitions

Logical Models are stored as XML documents in the LANSA root install directory (by default). These XML documents are not partition specific. Logical models are not stored in the LANSA internal database. Models can be shared across partitions and can even be shared across LANSA systems.

After starting Visual LANSA and logging into a LANSA partition, you could open any logical model saved on your system or on a network.

If you have started to build a logical model into a physical database, the build information initially displayed in the model is specific to the partition in which the model was last built.

It is possible to build the same logical model into many different partitions by simply opening the logical model and rebuilding it in the active partition.

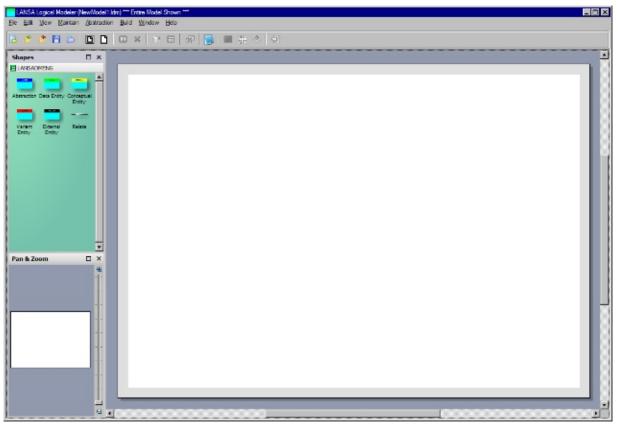
Before using the Logical Modeler in a partition, be sure to review the Visual LANSA partition requirements listed in the 2.1 Pre-Requisites.

2.3 Start the Logical Modeler

To open the Logical Modeler, select the LANSA editor *Tools* menu and choose the *Logical Modeler* option.

2.4 The Main Window

Once opened, the main Logical Modeler window will appear something like this:



It includes:

- 2.4.1 Menus
- 2.4.2 The Toolbar
- 2.4.3 The Model Diagram
- 2.4.4 The Document Stencil
- 2.4.5 Pan & Zoom Window.

You should also be familiar with 2.4.6 Performing Actions on Diagram Objects. Notice that the name of the model and any abstraction selected is displayed in the main window title bar. (The phrase Entire Model Shown indicates that no abstractions are active).

To access the user guide for the Logical Modeler, use the Help menu (pressing the F1 key when the main window is shown will display the Visio help text). Pressing the F1 key when a logical modeler dialog is shown (e.g. Create Data Entity) will show help for that dialog.

2.4.1 Menus

All actions available with the modeler can be executed from the menus on the modeler's main window. An option may or may not be available depending on the state of the model diagram.

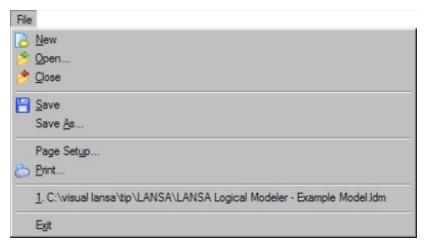
Note that an icon may be displayed to the left of a menu item. These icons are also available from the toolbar.

The menus on the main window are:

- File Menu
- Edit Menu
- View Menu
- Maintain Menu
- Abstraction Menu
- Build Menu
- Window Menu
- Help Menu

File Menu

The File menu is used to perform administration-type actions on the model. These actions include creating a new model, opening an existing model, saving a model and printing a model. It appears like this:



Edit Menu

The Edit menu is used to perform actions on entities, relationships and abstractions.

These actions include the Change and Delete commands (applicable to all four object types), Specify Relationship and the Entity Elements and User Views options (applicable only to certain types of entity). It appears like this:

Edit	
Chan	The second s
	ify <u>R</u> elationship
🤒 Entity 🔝 User	v Elements ⊻iews
Mote	S
Unat	tached Elements

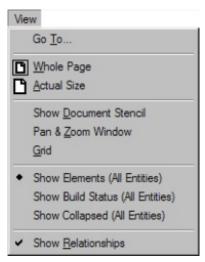
View Menu

The View menu provides options which affect the visual presentation of the diagram.

Options such as Actual Size and Whole Page can be used to enlarge or reduce the size of the diagram to aid readability. The Show Relationships options can be used to control whether relationship lines are shown on the diagram.

In addition, the Show Elements, Show Build Status and Show Collapsed options control the type of view of the model shown by the diagram. These views can be set for the entire model or for selected entities only.

The View menu appears like this:



Maintain Menu

The Maintain menu is used to access the Data Types used by element definitions. It appears like this:

Maintain <u>D</u>ata Types...

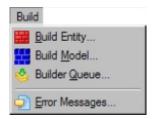
Abstraction Menu

The Abstraction menu is used solely for performing actions on Abstractions. When an abstraction is selected in the model, it can be activated or deactivated and it can have entities added or removed. It appears like this:

Abstraction
Activate
Deactivate
Specify Objects
Add Objects
<u>R</u> emove Objects

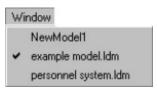
Build Menu

The Build menu is used to support the features associated with building your model. Build error messages can be accessed, an individual entity or the entire model build can be initiated, and the Builder Queue can be accessed. It appears like this:



Window Menu

The Logical Modeler allows for many data models to be open simultaneously. The Window menu allows you to switch between models that are currently open. It may appear something like this:



Help Menu

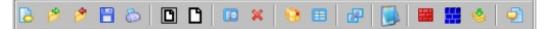
The Help menu provides access to this User Guide. It appears like this:

Help	
Contents	

2.4.2 The Toolbar

The Toolbar provides access to the most commonly used modeler commands. For example, you could Open Model, Save Model, Show Diagram Actual Size, Change, Delete, Build Model, etc.

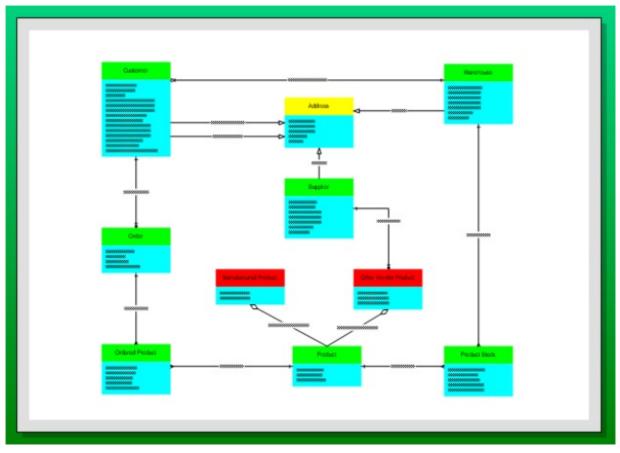
A toolbar button may or may not be enabled depending on the state of the model diagram. The toolbar appears something like this:



Moving the mouse pointer over a toolbar button will display the action associated with the button.

2.4.3 The Model Diagram

The main portion of the Logical Modeler window contains the diagram of your model. It shows either the entire model (as shown below), or a portion of the model as dictated by the different viewing options (as described in the 2.4.5 Pan & Zoom Window).



New objects are created by dragging shapes onto the diagram from the Document Stencil (as described in 2.4.4 The Document Stencil).

For the most part, maintenance of existing model objects is initiated by selecting shapes on the diagram and performing actions against them.

Shapes can be moved around the diagram to give a desired layout. This layout is preserved when the model is saved and reloaded.

2.4.4 The Document Stencil

The document stencil can be used to create new model objects. In addition, it acts as a legend for the model diagram. The types of objects shown in the diagram can be referenced to the types shown here. It appears something like this:



New objects can be created by dragging and dropping shapes from the stencil onto the model diagram. The objects available are:

- Data entity
- Conceptual entity
- Variant entity
- External entity
- Abstraction and
- Relationship connector.

For details of these object types, refer to 5.1 Entity Types, 8. Abstractions, and 7.7.1 Create a Relationship.

The document stencil can be anchored in the main window or it can be a floating window. When displayed as a floating window, it can be resized and positioned anywhere on the desktop.

Note that all objects available in the document stencil can also be created by right-clicking on a blank portion of the model diagram and selecting the appropriate option.

2.4.5 Pan & Zoom Window

The Pan & Zoom window is used to navigate within your model. It shows an image of the entire model and may appear something like this:

Pan & Zoom 🛛 🗆 🗙

Using the left mouse button, the zoom view (indicated by a red border) can be moved, resized or redrawn. The main diagram reflects the portion of the model bound by the red border.

Note that the current size and position of the Pan & Zoom window is saved when the model is saved and reinstated when the model is reopened.

The Pan and Zoom window can be anchored in the main window or it can be a floating window. When displayed as a floating window, it can be resized and positioned anywhere on the desktop.

2.4.6 Performing Actions on Diagram Objects

To perform an action on a diagram object (an entity, relationship or abstraction), it must first be selected.

To select an object, use a single click of the left mouse button on the object's shape. Once selected, the shape will have selection handles in the form of gray padlock images surrounding its border as shown in this example of the Customer entity:



To perform an action against the object:

- right-click the object and select an option from its pop-up menu, or
- select an option from one of the main window menus, or
- click a toolbar button that corresponds to the action you wish to perform.

Note: The Change, Delete and Diagram View options can be performed on more than one object simultaneously. To select multiple objects, hold down the Shift key and click on the desired shapes. When executed, the Change, Delete and Diagram View functions will be executed for all selected objects.

3. Models

A data model is a structured representation of data important to your business. It uses Elements, Entities and Relationships to translate business requirements into a graphical and textual model that can be understood regardless of the underlying database.

This section describes how to:

- 3.1 Creating a New Model
- 3.2 Opening a Model
- 3.3 Saving a Model
- 3.4 How Models are Stored
- 3.5 Switching Between Open Models
- 3.6 Model Views
- 3.7 Changing the Size of the Model Diagram
- 3.8 Navigating the Model Diagram
- 3.9 Positioning to an Entity or Abstraction on the Model Diagram
- 3.10 Printing the Model Diagram
- 3.11 Maintaining Unattached Elements
- 3.12 Adding Model Notes
- 3.13 Building the Model

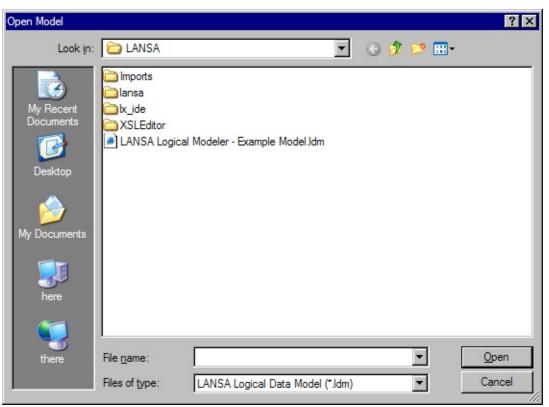
3.1 Creating a New Model

Select the File menu and choose the *New* command or click the **b** button on the main toolbar.

A blank model diagram will be displayed. Note that already opened models are accessible from the Window menu.

3.2 Opening a Model

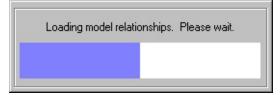
To open a previously saved model, select the File menu and choose the *Open*... command or click the solution on the main toolbar. The Open Model dialog is displayed:



By default, LANSA models are stored in the LANSA directory of your LANSA installation. Any models which exist in this directory will be displayed in the Open Model list.

In addition, the last nine models that have been opened appear in the File menu. To re-open one of these models, simply select the corresponding menu option.

Once a model has been selected to be opened, a progress bar, showing the different phases and how much of the load has taken place, is displayed while the model is loading, It appears like this:



Once the model has loaded, it will be shown in the main diagram. The model

view settings are remembered from when the model was saved.

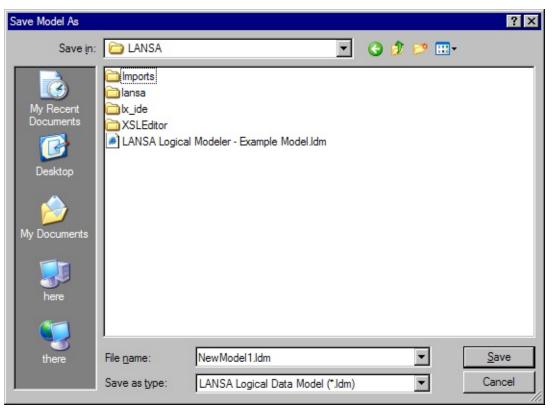
Note that the model is saved as an XML file and as a Visio document. If any errors are found in the model's XML, it will not load and a message explaining the error will be displayed. This should only occur if you have manually altered the XML and used invalid syntax or have manually removed shapes from the Model Visio document. It is strongly recommended that you do not manually alter model XML.

3.3 Saving a Model

To save a model:

- Use the 🖻 button on the main toolbar. If the model has not previously been saved, the Save Model As dialog will be displayed.
- Select the File menu and choose the *Save* command. If the model has not previously been saved, the Save Model As dialog will be displayed.
- Select the File menu and choose the *Save As...* command if you wish to save an existing model with a different name.

If the model has not previously been saved, or if you have selected the *Save As*... command, the Save Model As dialog is displayed:



Once the model has been saved, the model diagram will be redisplayed. The save will include information about the diagram such as the model view and the active abstraction.

3.4 How Models are Stored

When a model is saved, it is stored as XML and as a Visio document. The name of the XML file to which it is saved has the format:

x.ldm

Where x is a name supplied by you and .ldm is the standard suffix for LANSA data models.

The name of the Visio document to which the model is saved has the format: x.ldm.vsd

Where x is a name supplied by you, .ldm is the standard suffix for LANSA data models and .vsd is the standard suffix for Visio documents.

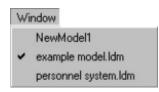
By default, models are saved in the LANSA directory of your LANSA installation. However, you may choose where to save the model when using the Save Model As dialog.

Once a model has been saved, it can be moved around or shared with other users easily, as it is completely independent of the internal database of your LANSA installation.

Note that, when loading a model, or sharing it with other user, only the model XML file is required. However, if the Visio document is not in the same location as the XML file, the model will take longer to load.

3.5 Switching Between Open Models

The Logical Modeler allows more than one model to be open at a time. A list of open models is displayed in the Window menu as shown in this example:



The model currently displayed has a tick to the left of its name. To switch to another model, simply select its name from the menu. The diagram will switch to the selected model.

The name of the active model is displayed in the title bar of the main window.

3.6 Model Views

There are three views available for entities in the model diagram:

- Elements view
- Build Status view
- Collapsed view.

The Elements view shows element names in the entity shape as shown in this Customer entity example:



The Build Status view displays the status of the three build phases for the Entity (Fields, File and Relationships) as shown in this Customer entity example:

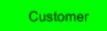


Each phase of the build may display four different statuses:

- Not yet built
- Required
- Built
- Failed

The Required, Built and Failed statuses are followed by the date and time at

which that status was assigned to the entity (i.e. either as the result of a build of the entity or as the result of a related entity having been changed or built). The Collapsed view simply displays the names of all entities in the model.



There are three options available from the View menu to control which view is displayed:

- Show Elements
- Show Build Status
- Show Collapsed

The View options are context-sensitive. When no diagram objects are selected, each of the options will change the view for the entire model. To denote this, the text '(All Entities)' is displayed next to each option.

When one or more entities are selected, each of the options will change the view for those entities only. To denote this, the text '(Selected Entities)' is displayed next to each option.

When a model is initially created, the default view is the Elements view for all entities. To toggle between this view and the Build Status or Collapsed views for all entities, ensure no entities are selected and choose the *Show Build Status (All Entities) or Show Collapsed (All Entities) options* from the View menu.

To specify the view to be assigned to a select group of entities, select those entities and choose the view. For example, if the Elements view for all entities is shown, but you wish to look at the Build Status view for one or more individual entities, you select the required entity shapes and then select the *Show Build Status (Selected Entities)* option from the View menu.

Similarly, if the Build Status view for all entities is shown, but you wish to look at the Elements view for one or more individual entities, select the required entity shapes and then select the *Show Elements (Selected Entities)* option from the View menu.

A check mark is shown to the left of the view descriptions to indicate which view is active for the current selection.

When you save the model and reopen it, the view of the model at the time it was saved will be restored.

In addition, you can elect whether to show the lines on the diagram that represent relationships by checking or unchecking the Show Relationships option on the View menu. This option can be useful in simplifying the model view if many relationships are specified.

3.7 Changing the Size of the Model Diagram

The size of the model diagram refers to the size of the paper that the diagram would be printed on. By default, it is the size of a sheet of A4 or letter paper (depending on your system settings).

If your model is too large to fit all of its objects onto a sheet of A4, you can change the paper size by selecting File menu and choosing the Page Setup... option. This displays the standard Visio Page Setup dialog. Select the Page Size tab. The dialog will appear something like this:

Page Setup	×
Print Setup Page Size Drawing Scale	Page Properties Layout and Routing
Page size	
Same as printer paper size	
C Pre-defined size:	Printer Paper
Standard	
Letter: 11 in x 8.5 in	
O Custom size:	Drawing Page
297 mm × 210 mm	
Size to fit drawing contents	Printer paper: 297 × 210 mm (Landscape)
Page orientation	Drawing page: 297 × 210 mm (Landscape)
🔿 Portrait 💿 Landscape	Print zoom: None
2	Apply OK Cancel

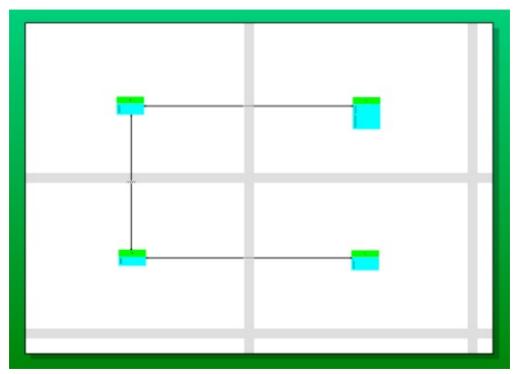
To change the size of the diagram, you can select and change any of the options displayed in this dialog.

Press *OK* if you wish to change the diagram size. The diagram will be redrawn appropriately. Note that object shapes may be repositioned due to the size change.

Press *Cancel* if you do not wish to change the size of the diagram.

Note also that if the page size is larger than the printer paper size (as defined on the Print Setup tab of this dialog), gray lines will be displayed on the diagram to indicate the number of pages the model will be printed on. For example, in the diagram below, the Page size is set as A2 and the printer paper size is set to A4. When the model is printed, four pages will be produced, corresponding to the

pages represented by the gray lines.



3.8 Navigating the Model Diagram

A representation of the entire model diagram is shown in the Pan & Zoom window while either the entire diagram or a portion of it is shown in the model diagram area.

To change what is shown in the model diagram:

- Use the two page view functions: *Whole Page* and *Actual Size* on the View menu.
- Change the zoom view, indicated by a red border, in the Pan & Zoom window. This view can be moved, re-sized or re-drawn by clicking and holding down the left mouse button. (Note that the Pan & Zoom window does not change. It still continues to show the entire model.)
- Use the vertical and horizontal scroll bars that border the model diagram.

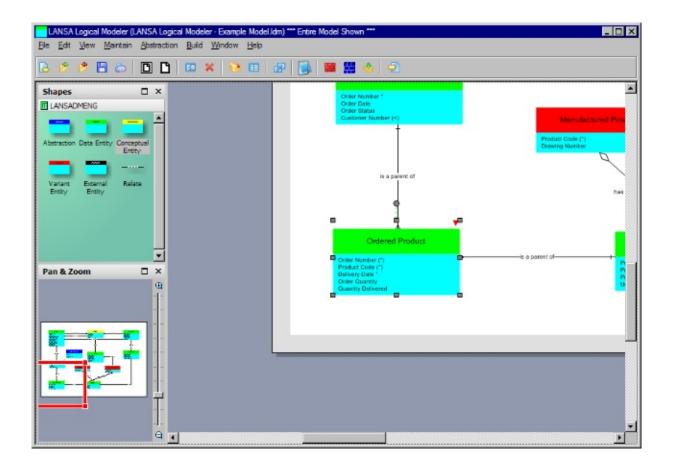
3.9 Positioning to an Entity or Abstraction on the Model Diagram

To go straight to an Entity or Abstraction on the model diagram, select Go To... from the View menu. A dialog containing a list of all model entities and abstractions will be displayed:

🗧 Go To		_ 🗆 ×
Object		
Address		
Customer		
Manufactured Product		
Order		
Order Processing		
Ordered Product		
Other Vendor Product		
Product		
Product Stock		
Supplier		
Warehouse		
•		
		1
	Go To	Cancel

To go to the required shape on the diagram, double-click a list item, or select a list item and press the Go To button.

The modeler will change the diagram to actual size, select the shape and position it in the centre of the display, as shown below:



3.10 Printing the Model Diagram

To print the model diagram, click the button on the main toolbar, or select the File menu and choose the *Print*... command. The Print dialog is displayed:

×
Properties
🔽 Color as <u>b</u> lack
🗖 Print to fi <u>l</u> e
Copies
Number of copies:
1 2 3 3
OK Cancel

This is the default Visio print dialog, and behaves as a standard Windows print dialog.

Note that each open model corresponds to a single page. Printing all pages when more than one model is open will print all models. To be sure that only the currently displayed model is printed, select the *Current* page option before printing.

3.11 Maintaining Unattached Elements

When modeling, elements may be known, but the entities to which those elements should belong is not known. Also, when an entity is deleted from the model, all of the elements which were defined in the entity still exist but are no longer grouped into a specific entity. The Logical Modeler offers the ability to maintain these elements and to create more elements without immediately attaching them to entities.

To maintain Unattached Elements directly, select the *Edit* menu and choose the *Unattached Elements* ... option. The Maintain Unattached Elements window is displayed:

Idress Line 2 Name Alphanumeric (25)	ement name Address Line 1	Data type Name	Field type/length Alphanumeric (25)	Field name
der Number Integer (4)	Address Line 2	Name	Alphanumeric (25)	
	Order Number	ID Number	Integer (4)	

A list of currently unattached elements is displayed, alphabetically, below the toolbar.

The following information is shown for each element in the list:

- its build status (a green tick if the element has been built, a red cross if the element is required to be built)
- its name
- a Notes image, if the element has notes attached
- its data type, if it has one,
- its field type and length, or the type and length of the data type, if used
- its field name

From this window:

• A new element can be created by clicking the 🗟 button on the toolbar.

- An element can be changed, by double-clicking it in the list, or by selecting it and clicking the **1** toolbar button.
- An element can be deleted by selecting it and clicking the 🔀 toolbar button.
- An element's notes can be maintained by selecting it and clicking the Determination toolbar button.

For information about these options, refer to the 6.4 Working with Elements.

Unattached elements can be attached to entities at a later stage by dragging and dropping them from this window into an Entity Elements window. Similarly, attached elements can be unattached by deleting them from an Entity Elements window or by dragging them and dropping them from an Entity Elements window into the Unattached Elements window. Refer to 5.6.6 Transferring Elements for further information.

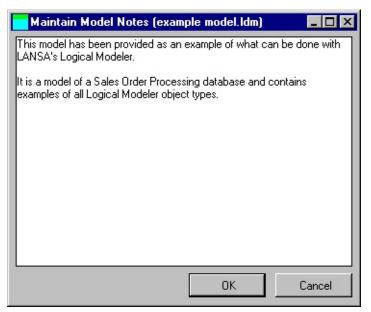
3.12 Adding Model Notes

The notes facility for models is merely a documentation tool for use when constructing a data model. Unlike element notes, which are transformed into field help text when the element is built, model notes are not carried through from the model to the repository.

To maintain a model's notes:

- right-click on an empty part of the diagram and select the *Notes... option from the menu*,
- ensure that no diagram objects are selected and choose the *Notes*... option from the Edit menu, or
- ensure that no diagram objects are selected and click the D button on the toolbar.

The Maintain Model Notes dialog is displayed:



The standard LANSA special help text characters may be used in the notes. For example, you can display help text underlined, highlighted and so on.

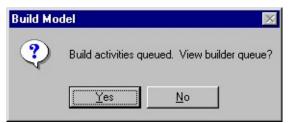
Once the notes have been maintained, press the *OK* button to update them. If you do not wish the notes to be updated, press the *Cancel* button.

3.13 Building the Model

To build a model, use the **E** button on the main toolbar or select the Build menu and choose the *Build Model*... command. Alternatively, right-click on an empty part of the diagram and choose the Build Model... option. The Build Analysis function will be performed.

The Build Analysis function can operate in two ways. If there is no abstraction currently active, the entire model is analyzed and may be built. If an abstraction is active, only the entities within the abstraction will be analyzed. Refer to 8. Abstractions for more information.

If none of the entities being analyzed have previously been built, the analysis function will submit build activities to the Builder Queue for all entities and a message box is displayed:



Select *Yes* to display the Builder Queue immediately, otherwise select *No* to return to the model diagram. For more information about managing the build process with the Builder Queue, refer to 10. Building a Model.

If the entities being analyzed are in a partially or fully built state, a window is displayed:



If you wish to build only the entities that are required to be built, select the Entities requiring build radio button and press the OK button or the Enter key.

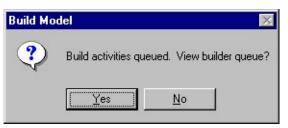
If you wish to build all entities, regardless of their current build status, select the All entities radio button and press the OK button or the Enter key.

Press the Cancel button or the Escape key if you do not wish to build any entities.

If you have selected the Entities requiring build function but all entities are already built, a message box will be displayed:



If the analysis function does find entities that need building, or you selected the All entities option, build activities are submitted to the Builder Queue and a message box is displayed:



Select *Yes* to display the Builder Queue immediately, otherwise select *No* to return to the model diagram. For more information about managing the build process with the Builder Queue, refer to 10. Building a Model.

Note: It is possible to build individual entities by selecting an entity and clicking the **button**. For more information about building entities individually, refer to **5.6.9 Building an Entity**.

4. Data Types

A data type is basically a link to a corporate or standard field definition in the LANSA Repository. For example, the modeler has a data type called "Name" which refers to the field STD_NAME in the repository. When you create a "Customer Name" element, you would specify that it has a data type of "Name". Likewise, when you create another element called "Supplier name", it will also access the data type "Name". This ensures that you have one corporate standard for all "names" in your model.

If you require a "Company name" which has completely different rules and properties, then you have identified a new data type. A standard or corporate definition should be defined to the repository, and a new data type should be created.

The modeler supports the concept of a corporate repository. As you create new corporate data types, you will be adding them to the list of element data types.

Data types should not form a complete duplicate of your repository. You should not create a new data type for each element in your model. Data types represent global types of data for your models, applications and databases.

If you wish to use the data typing facility, one of the most common changes you will make to the LANSA environment is the maintenance of data types in the Logical Modeler.

Elements that refer to data types for their definition inherit the data type field's physical characteristics, including validation rules and help text, when the repository definition for an element is built.

The Logical Modeler is shipped with some default data types. These shipped data types all refer to the standard LANSA fields that begin with the characters STD_. (Refer to 2.1 Pre-Requisites.)

When you work with Data Types, you will use these modeler facilities:

- 4.1 Maintaining Data Types
- 4.1.1 Creating a Data Type
- 4.1.2 Changing a Data Type
- 4.1.3 Deleting a Data Type

4.1 Maintaining Data Types

To maintain data types, select the Maintain menu and choose the *Data Types...* option. The Maintain Data Types window is displayed:

Maintain Data Types			_ 🗆 2
🔊 👌 🗈 🗙			
Data/field type	Repository field	Field type/length	-
Amount	STD_AMNT	Packed (9,2)	
Boolean	STD_BOOL	Alphanumeric (5)	
Code	STD_CODE	Alphanumeric (3)	
Code (Long)	STD_CODEL	Alphanumeric (6)	
Code (Short)	STD_CODES	Alphanumeric (1)	
Count	STD_COUNT	Packed (7,0)	
Date	STD_DATE	Signed (6,0)	
Day	STD_DAY	Signed (2,0)	_
Description	STD_DESC	Alphanumeric (30)	
Description (Long)	STD_DESCL	Alphanumeric (40)	
Description (Short)	STD_DESCS	Alphanumeric (15)	
Hour	STD_HOUR	Signed (2,0)	
ID Number	IDINTEGER	Integer (4)	
ID Number (Long)	STD_IDNOL	Packed (15,0)	
ID Number (Short)	STD_IDNOS	Packed (3,0)	

A list of defined data types is displayed, alphabetically, below the toolbar. The following information is shown for each data type in the list:

- its name
- the LANSA Repository field it uses
- the field type and length.

From this window, you can:

- 4.1.1 Creating a Data Type
- 4.1.2 Changing a Data Type
- 4.1.3 Deleting a Data Type.

4.1.1 Creating a Data Type

To create a new data type, click the **b** toolbar button. The Create Data Type dialog is displayed:

Create Data Typ	e		
Data/field type Repository field		<u></u>	
		ОК	Cancel

The Data Type must be given a name. It can be up to 20 characters in length of mixed case, and must not already exist.

A Repository field name must be specified. Enter the name of a field that exists in the repository, or click the 🖃 button to select a field from a list.

Be very careful when selecting a field from the LANSA Repository. You should review the field definition, its help text and validation rules as the modeler uses this information when creating new fields based on this data type. For example, the selected Repository field should not have any simple logic checks using the name of the field as these type of rules cannot be copied to another field.

Click the *Cancel* button if you do not wish to create a data type.

Click *OK* to perform validation and create the data type. The dialog will be redisplayed for you to continue create data types. Press *Cancel* to return to the data types list.

4.1.2 Changing a Data Type

To change an existing data type, double-click its entry in the Maintain Data Type list, or select its entry and click the ¹⁰ toolbar button. The Change Data Type dialog is displayed:

Change Data Ty	pe		
Data/field type	Description		
Repository field	STD_DESC		
		OK	Cancel

The name of the Data Type cannot be changed. (You must delete the existing data type and add using the new name.)

A repository field name must be specified. It must be the name of a field that exists in the repository, or use the 💷 button to select a field from a list.

Click the *Cancel* button if you do not wish to change the data type.

Click *OK* to perform validation and update the data type.

Note: If you change a data type, you should manually rebuild all elements which use the changed data type.

4.1.3 Deleting a Data Type

To delete a Data Type, select its entry in the Maintain Data Type list and click the 💌 toolbar button. The data type will be removed from the list.

Note: The data type will not be deleted if it is used with an element in a currently open model. An error message will be displayed if to indicate why the data type cannot be deleted.

5. Entities

Entities are the main focus of your development in the Logical Modeler. They are the starting point for most model development.

An entity can be anything that a user works with in their business. For example, in an order entry application, a Product, a Customer or an Order may be an entity.

For each entity, you can:

- define elements,
- define relationships to other entities,
- add notes,
- build the entity, and
- review build error messages.

Each entity must be assigned a type. This is very important in determining how entities interact with each other via relationships, and how they are implemented in the repository when the model is built.

This information is described in:

- 5.1 Entity Types
- 5.2 Entities and Elements
- 5.3 Entities and Relationships
- 5.4 Entities and LANSA
- 5.5 Working with Entities
- 5.6 Maintain Entity Elements

5.1 Entity Types

An entity's type determines:

- What type of relationships an entity may have with other entities and
- if and how the entity will be implemented in the repository.

For example, a Conceptual Entity can only be used with Include relationships. Also, a Conceptual Entity is not implemented as a file in the repository. Determining the correct entity type is very important to your model.

The Logical Modeler has the following types of entities:

- Data Entity
- Conceptual Entity
- Variant Entity
- External Entity.

Note: Abstractions are not entities. They are objects used to group entities.

Data Entity

A Data Entity stores information. It is the most commonly used type of entity. For example, a Customer is a data entity.

A Data Entity has very few relationship restrictions. It can have an identifying element. It is implemented as a file in the LANSA Repository.

Conceptual Entity

A Conceptual Entity has meaning to the user but does not store information. For example, Address or Date might be modeled as a Conceptual Entity.

A Conceptual Entity is restricted to being the target of an Include relationship. It cannot have an identifying element. It is implemented as a set of fields in the repository. It cannot have user views.

Variant Entity

A Variant Entity is similar to a Data Entity in that it stores information. It is used in situations where elements are dependent on the value of another element. For example, Manufactured Product and Supplied Product are variant entities used with the Product data entity.

A Variant Entity is initially restricted to be the target of a Variation relationship. However, once defined, it can be the subject of a Join relationship. It cannot be assigned its own identifying element as its key is identified by the variation relationship. It is implemented as a file in the LANSA Repository.

External Entity

An External Entity is a representation of an existing physical or logical file in the LANSA Repository. It can be used in relationships with Data and Variant entities in order to extend an existing database.

An External Entity can be the Parent entity in a Parent/Child relationship, the target entity in a Join relationship and the source entity in a Variation relationship. Its elements cannot be maintained.

Note that, during the Build process, the database file that corresponds to the External Entity is not modified in any way. However, Data and Variant entities that are related to the External Entity may have appropriate Access Routes, Logical Views and Referential Integrity Validation Rules created for them.

5.2 Entities and Elements

An Entity's type directly influences its elements. The entity type will determine the other entity types to which it can be related and whether an identifying element can be specified.

Because the entity type determines how an entity is implemented in the repository, rules must apply to what an entity contains:

- Conceptual entities cannot be assigned identifying elements (remember, conceptual entities are not implemented as files).
- Variant entities cannot be assigned an identifying element. It receives an identifying element as the result of a Variation relationship from a Data or External Entity.
- External entities cannot have their elements maintained (remember, the underlying physical or logical file definition is not modified by the modeler).

5.3 Entities and Relationships

The Logical Modeler only allows meaningful relationships to be defined for a given source entity type and target entity type. For example, a Parent/Child relationship can only be used by Data and External entities. An Includes relationship can only use Conceptual entities as the target entity. The Modeler helps to validate the model and your design based on these relationship rules. For more details about these rules, refer to 7.4 Relationships and Entities.

5.4 Entities and LANSA

Entity type determines what object is implemented in the LANSA Repository when the model is built.

An entity type will influence:

- Whether repository object are built for the entity,
- the physical object implemented and,
- the type of validation rules, access routes and logical files implemented, based on its relationships with other entities.

Data and Variant entities are implemented as physical and logical files.

Conceptual entities are implemented as fields. Conceptual entities cannot have user views.

External Entities already exist in the LANSA repository and are not implemented as database files. However, Data and Variant entities that are related to the External Entity may have appropriate Access Routes, Logical Views and Referential Integrity Validation Rules created for them.

5.5 Working with Entities

You will work with entities using 2.4.4 The Document Stencil and by 2.4.6 Performing Actions on Diagram Objects.

The following dialogs allow you to:

- 5.5.1 Create a Data, Conceptual or Variant Entity
- 5.5.3 Create an External Entity
- 5.5.4 Change a Data, Conceptual or Variant Entity
- 5.5.5 Delete an Entity

5.5.1 Create a Data, Conceptual or Variant Entity

Creating a Data, Conceptual or Variant entity is as simple as dragging one of these entity shapes from the Document Stencil onto the model diagram:



Data Entity Conceptual Entity Variant Entity

Alternatively, right-click on an empty part of the diagram and choose the appropriate Create option from the menu.

An appropriate dialog will be displayed, allowing you to specify entity details. For example, the following dialog would be displayed for creating a data entity:

Create Data Entity			
Entity name			
Physical file name			
		OK	Cancel

Regardless of the type of entity, it must be given a name. This can be up to 20 characters in length, of mixed case, and should be a unique name within the model.

For Data and Variant entities, which are implemented in the repository as files, a physical file name may be optionally specified. This will be the name of the physical file in the repository when the entity is built. The file name can be up to 10 characters in length if your Visual LANSA installation has the *RPGIV generation option turned on (refer to the Visual LANSA User Guide for more information). If it is turned off, the file name can be up to 8 characters in length.

Note that if a physical file name of more than the maximum length allowed is specified, relationship and user view logical view names must be specified manually. For more information about specifying physical file names, refer to 5.5.2 About Entity Physical and Logical File Names, below.

A physical file name cannot be specified for a Conceptual entity.

Note that, at this point, the entry of a physical file name is optional – it is not required for the purposes of constructing the data model. However, it must be

specified when building the entity. A physical file name can be entered using the Change facility. If no physical file name is specified when building the entity, a build error will be generated.

Press the *OK* button or Enter key to create the entity. The entity shape will be updated with the entity name. Note that the shape can now be moved anywhere on the diagram.

Press the *Cancel* button or *Escape* key if you do not wish to create the entity. The entity shape will be deleted from the diagram.

5.5.2 About Entity Physical and Logical File Names

As discussed previously, a physical file name may be up to 10 characters in length if the *RPGIV setting is enabled for your installation (refer to the Visual LANSA User Guide for more information). If it is not enabled, a file name can only be up to 8 characters in length.

Logical views are created by the modeler as a result of relationships between entities and of the User Views defined for entities.

The Logical Modeler provides a facility to automatically generate logical view names based on the physical file name of the entity to which they belong. These names are in the format xxxxxxnn, where xxxxxxx is the physical file name of the entity and nn is a unique, sequential number starting at 1.

Therefore, in order for the modeler to be able to automatically generate logical view names, a physical file name of up to the maximum length allowed only may be specified.

If the physical file name is more then the maximum number of characters allowed, logical view names must be specified manually. This is done via the Create- & Change Relationship and Specify User View dialogs.

For more information on specifying logical view names, refer to the following:

- 7.7 Work with Relationships
- 9. User Views
- 10. Building a Model

5.5.3 Create an External Entity

To create an External Entity, drag the following shape from the Document Stencil onto the model diagram:



Alternatively, right-click on an empty part of the diagram and choose the appropriate Create External Entity... option from the menu.

The Import File Definition dialog will be displayed:

Import File Defi	nition		
File/library			
		OK	Cancel

A File and its Library name must be specified. The file must exist in the LANSA Repository. Press the prompt button to the right of the Library name in order to select a file from a list of files in the repository. The file may be a Physical or Logical file. If the file is a Physical file, it must have at least one primary key field defined.

Press the *OK* button or Enter key to create the entity. The entity shape will be updated with the file name and the fields of the file.

Press the *Cancel* button or *Escape* key if you do not wish to create the entity. The entity shape will be deleted from the diagram.

5.5.4 Change a Data, Conceptual or Variant Entity

To change an entity's details, select its shape on the diagram and:

- select the *Change*... command from the entity's right-click menu, or
- select the *Change*... command from the Edit menu, or
- click the 🔟 button on the toolbar.

Note that External entities cannot be changed.

An appropriate dialog will be displayed according to the type entity you are changing. For example, the following dialog would be displayed for a data entity:

Create Data Entity			
Entity name	Customer		
Physical file name	TBCUST		
		OK	Cancel

The entity name may be changed if required. It can be up to 20 characters in length, of mixed case, and should be a unique name within the model.

If a physical file name has not yet been specified, it may be specified here. It will become the name of the file in the repository when the entity is built. It can be up to 8 or 10 characters in length, depending on the *RPGIV setting of your Visual LANSA installation, should follow normal file naming standards and must be unique within the model.

Note that if a physical file name of more than the maximum number of characters allowed is specified, relationship and user view logical view names must be specified manually. For more information about specifying physical file names, refer to 5.5.2 About Entity Physical and Logical File Names.

If a physical file name has been specified and the entity has been previously built it cannot be changed. This ensures integrity between entities when building the data model.

Press the *OK* button or Enter key to change the entity. The entity shape will be updated with the new entity name if it was changed.

Press the *Cancel* button or *Escape* key if you do not wish to change the entity.

5.5.5 Delete an Entity

Deleting an entity deletes the selected entity and any relationships between it and other entities.

To delete an entity, select its shape on the diagram and:

- select the *Delete*... command from the entity's right-click menu, or
- select the *Delete*... command from the Edit menu, or
- click the ຶ button on the toolbar.

You will be asked to confirm the entity's deletion.

If the selected entity includes elements as the result of a relationship with another entity, those elements are detached from the selected entity. Similarly, if another entity contains elements from the selected entity as the result of a relationship, those elements are detached from the entity.

Any elements in a deleted Data, Conceptual or Variant entity become available as Unattached Elements for later use if required (refer to 3.11 Maintaining Unattached Elements).

In addition, the build statuses of the selected Data, Conceptual or Variant entity and any related entities are reset appropriately if they have been previously built.

5.6 Maintain Entity Elements

To maintain an entity's elements, select the entity's shape on the diagram and:

- select the *Elements*... option from the entity's right-click menu, or
- select the *Entity Elements*... option from the Edit menu, or
- click the 随 button on the toolbar.

The Maintain Entity Elements dialog, which consists of a toolbar and a list of elements that belong to the entity, is displayed:

2 🔒 🛛 🗶 🖓 🖑	8 🔰 🗹		
Name	Data type	Field type/length Field name	From entity
🕜 Customer Number		Packed (9,0)	
Customer Name		Alphanumeric (50)	
Credit Limit	Amount	Packed (9,2)	
(>) Delivery Address Line 1	Name	Alphanumeric (25)	Address
(>) Delivery Address Line 2	Name	Alphanumeric (25)	Address
(>) Delivery Address Line 3	Name	Alphanumeric (25)	Address
(>) Delivery Post Code	Zip/Post Code	Alphanumeric (5)	Address
(>) Delivery Country	Name	Alphanumeric (25)	Address
(>) Billing Address Line 1	Name	Alphanumeric (25)	Address
(>) Billing Address Line 2	Name	Alphanumeric (25)	Address
(>) Billing Address Line 3	Name	Alphanumeric (25)	Address
(>) Billing Post Code	Zip/Post Code	Alphanumeric (5)	Address
(>) Billing Country	Name	Alphanumeric (25)	Address
(<) Default Warehouse Code		Alphanumeric (10)	Warehouse

Elements shown here have either been explicitly defined as belonging to this entity or are shown as the result of the entity's relationships with other entities.

Note that an External Entity's elements cannot be maintained in any way.

The following information is shown for each element in the dialog:

- Its build status (a green tick if the element has been built, a red cross if the element is required to be built)
- A symbol indicating whether the element is an identifier or is shown as the result of a relationship with another entity. The symbols in use are:
- (>) for includes
- (<) for joins
- a gray key for parent identifiers
- a gold key for identifiers.

- Its Name,
- A notepad icon, if the element has notes
- Its data type, if it has one
- Its field type and length, or the type and length of the data type if it has one
- Its field name
- Its originating (or from) entity, if it is shown as the result of a relationship with another entity.

This dialog allows you to:

- Create elements for the entity.
- Change elements in the entity.
- Delete elements from the entity.
- Set an identifying element, as described in 5.6.4 Setting an Identifying Element.
- 5.6.5 Re-sequencing Elements.
- Maintain element notes.
- View element 5.6.8 Validation and Build Error Messages.

Changes to the entity's elements and its relationships with other entities will be reflected in the model diagram once this dialog is closed.

You should also be familiar with how to 5.6.6 Transferring Elements and 5.6.9 Building an Entity.

5.6.1 Creating Elements

Refer to 6.4.1 Create an Element.

5.6.2 Changing Elements

Refer to 6.4.2 Change an Element.

5.6.3 Deleting Elements

Refer to 6.4.3 Delete an Element.

• **Note:** Deleting an element that has been inherited via a relationship will delete the entire relationship with the opposing entity.

5.6.4 Setting an Identifying Element

An identifying element is used to identify a specific occurrence in the entity. For example, a Customer entity may have an identifying attribute of Customer Code. The Customer Code is used to uniquely identify a Customer.

The identifying element is set by selecting the element to be used as the identifying element and clicking the *identifying* toolbar button. You can change the identifying element by selecting another element and pressing the *identifying* button. If you use the *identifying* button with the current identifying element selected, it will be changed back to a standard element for the entity.

Note that more than one element may be necessary to identify an occurrence within an entity. In these cases, Parent/Child relationships supply inherited elements to make up the complete set of identifying elements. Inherited identifiers are denoted by a gray key symbol immediately to their left.

It is very important to remember to add identifying elements if you have relationships between entities. If entities are used in relationships and do not have identifying elements, the model build will fail with validation error messages.

If the entity is built into a database file, the identifying element will be used as part of the key to the file. The complete file key may be based on inherited elements as well as the identifying element.

Almost all entities will have an identifying element. If an entity has two or more parents, it does not require an identifying element.

Note that identifying elements are not permitted in Conceptual or Variant entities.

5.6.5 Re-sequencing Elements

Selecting an element and using the \bigcirc and \bigcirc toolbar buttons allows you to resequence the order of elements as they are listed in the dialog. The element sequence affects the sequence in which fields are built in the file.

If the element selected and moved is inherited from another entity as the result of a relationship, all inherited elements via that relationship will be moved.

If the entity is a Data or Variant entity, the sequence of the elements will determine the sequence of the fields in the physical file when the entity is built.

For a Conceptual entity, the sequence of the elements is reflected in any entity which inherits the elements via an Includes relationship. The sequence of inherited fields cannot be changed in the source entity. It can only be changed in the target Conceptual entity.

5.6.6 Transferring Elements

The Logical Modeler allows you to attach and detach elements to and from entities, and to transfer elements from one entity to another. Note that elements cannot be transferred to or from an External Entity.

If you have unattached elements that you wish to transfer to an entity, simply drag-and-drop the required elements from the Unattached Elements window (described in 3.11 Maintaining Unattached Elements) to the appropriate Entity Elements window. The following screen capture shows the Credit Limit element being dragged from the Unattached Elements window to the Customer Entity Elements window:

Name	Data type	Field type/length	Field name	From entity	
🗙 🥜 Customer Number	ID Number	Integer (4)			
x Customer Name		Alphanumeric (50)			
(>) Delivery Address Line 1	Name	Alphanumeric (25)		Address	
(>) Delivery Address Line 2	Name	Alphanumeric (25)		Address	
x(>) Delivery Address Line 3	Name	Alphanumeric (25)		Address	
x(>) Delivery Post Code	Zip/Post Code	Alphanumeric (5)		Address	
x(>) Delivery Country	Name	Alphanumeric (25)		Address	
x(>) Billing Address Line 1	Name	Alphanumeric (25)		Address	
x(>)Billing Address Line 2	Name	Alphanumeric (25)		Address	
x(>)Biling Address Line 3	Name	Alphanumeric (25)		Address	
x(>)Biling Post Code	Zip/Post Code	Alphanumeric (5)		Address	
x(>)Biling Country	Name	Alphanumeric (25)		Address	
x(<) Default Warehouse Code		Alphanumeric (10)		Warehouse	
× Credit Limit	Amount	Packed (9,2)			
	Maintain Unattached E	lements			
	🕫 👌 🗈 🗙				
	Element name	Data type		d type/length	Field nam
	x Credit Limit	Amount	Pac	cked (9,2)	

Similarly, elements can be transferred from one entity to another by draggingand-dropping them between the appropriate Entity Elements windows. The following diagram shows several elements being dragged from the Product entity to the Product Stock entity:

ne Warehouse Code Product Code	ID Number	Field type/tength Alphanumeric (10) Integer (4)	Field name	From entity Warehouse Product		
On-Hand Quantity Committed Quantity Back-Order Quantity	Quantity Quantity Quantity	Packed (7,0) Packed (7,0) Packed (7,0)				
Name		🚸 🕜 🚺 🧟 Data type ID Numbe		type/length	Field name	From entity
	Product Name Product Type	Name	Alph	anumeric (25) anumeric (10)		
				anumeric (2)		
	Init of Measure					
×	Init of Measure In-Hand Quantity Committed Quantity	Quantity Quantity		ved (7.0) ved (7.0)		

When dragging elements into an entity, they are removed from the source location and added to the end of the entity's existing elements. When detaching elements by dragging them to the Unattached Elements window, they are removed from the source location and added to the Unattached Elements list in alphabetical order.

Note that identifying elements and elements inherited from other entities cannot be transferred in this manner.

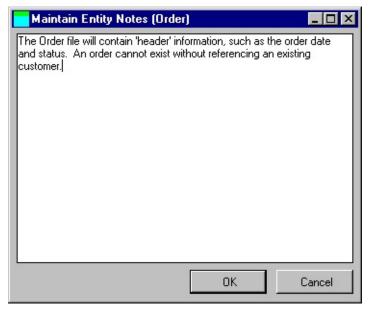
5.6.7 Entity Notes

The notes facility for entities exists as a documentation tool for use when constructing a data model. Unlike element notes, which are transformed into field help text when the element is built, entity notes are not carried through from the model to the repository.

To maintain an entity's notes, select the entity's shape on the diagram and:

- select the *Notes*... option from the entity's right-click menu, or
- select the *Notes*... option from the Edit menu, or
- click the 🗾 button on the toolbar.

The Maintain Entity Notes dialog is displayed:



The standard LANSA special help text characters may be used in the notes. For example, you can display help text underlined, highlighted, etc.

Once the notes have been maintained, press the *OK* button to update them. If you do not wish the notes to be updated, press the *Cancel* button.

On the model diagram, an asterisk (*) will appear after the entity name if it has notes attached.

5.6.8 Validation and Build Error Messages

Error messages are generated if validation errors are detected when a change is made to the entity or when part of the entity's build fails. If it has errors, an inverted red triangle will be shown above and to the right of the entity's shape, as shown below:

	Ţ
Customer	
Customer number *	
Customer name	
Credit limit	
Default Warehouse code (<)	
Delivery Address line 1 (>)	
Delivery Address line 2 (>)	
Delivery Address line 3 (>)	
Delivery Post code (>)	
Delivery Country (>)	
Billing Address line 1 (>)	
Billing Address line 2 (>)	
Billing Address line 3 (>)	
Billing Post code (>)	
Billing Country (>)	

To display an entity's error messages, select the entity's shape on the diagram and:

- select the *Error Messages...* option from the entity's right-click menu, or
- select the *Error Messages...* option from the Build menu, or
- click the 횓 button on the toolbar.

By default, all error messages relating to the build of the entity will be displayed, along with the date and time at which they were generated as in this example:



Use the check boxes at the bottom of the dialog to limit the messages displayed to only the Fields, File or Relationships phases of the entity build.

Press the *OK* button to close the dialog.

5.6.9 Building an Entity

To build an entity, select the entity's shape on the diagram and:

- select the *Build*... command from the entity's right-click menu, or
- select the *Build Entity*... command from the Build menu, or
- click the 📕 button on the toolbar.

The Submit Entity Build dialog is displayed.

Submit Entity Build	
Customer number	
Customer name	
Credit limit	
🖨 🗹 Build file	
Customer	
🖻 🗹 Build relationships	
Order	
Warehouse	
	1
OK	Cancel

You are given some flexibility over what phases of the entity build are performed. Depending on the current build status of the entity, the check boxes for the Fields, File and Relationships build may or may not be checked or protected, allowing you to select or de-select build phases or parts therein. Building fields:

- Those fields which have never been built are automatically selected and cannot be de-selected they are required to be built when you build the entity's fields. If the entity has not previously been built, all fields are selected and cannot be de-selected they must all be built the first time the entity is built. These fields are indicated by a gray checkbox.
- If a field has been changed since the last build, it is automatically selected but can be de-selected if you do not wish to build it at this time. These fields are indicated by a ticked black checkbox.
- If a field has not been changed since the last build, it is not selected but can

be selected for rebuild if you wish.

Building the file:

• If the file has never been built, or has changed since it was last built, the file is automatically selected. It can be de-selected if you do not wish to build it at this time.

Building relationships:

• If the file has never been built, or if relationships affecting the entity have been added or changed, all relationships are automatically selected. They can be de-selected if you do not wish to build relationships at this time.

If you do not wish to perform any build activity for the entity, press the *Cancel* button.

Press the *OK* button to submit the selected build activities to the Builder Queue. The Submit Entity Build dialog is displayed:



Select *Yes* to display the Builder Queue, from where builds can be executed or *No* to return to the model diagram.

For more information regarding building entities, refer to 3.13 Building the Model.

6. Elements

Elements are used to define the characteristics or attributes of an entity.

For example, a Customer entity might have elements of Customer Name, Customer Code, Credit Limit etc.

Elements can be created:

- within an entity, or
- as unattached elements that can be attached to an entity at a later stage.

Elements in the Logical Modeler follow the rules of Chen E-R Data Models. For instance, the following rules apply:

- An entity may have only one identifying element.
- An entity can inherit elements as the result of relationships with other entities.
- An element may be defined only once to a model.
- If an element exists in more than one entity, it must do so by means of a relationship.

This information is described in:

- 6.1 Elements and Entities
- 6.2 Elements and Relationships
- 6.3 Elements and LANSA
- 6.4 Working with Elements.

6.1 Elements and Entities

Data, Conceptual and Variant entities may have elements.

Conceptual entities do not get transformed into database files in the repository, and so cannot have identifying elements.

Variant entities can only inherit their identifying element(s) from another entity as the result of a Variation relationship.

External entities are defined in the repository and cannot have their elements maintained in any way.

If an element is not part of an entity, it is an unattached element.

Abstractions are not entities and may not contain elements.

6.2 Elements and Relationships

Relationships have a direct impact on the elements within an entity. The Logical Modeler follows the rules of Chen E-R Data Models. Of particular importance are the following rules:

- An entity will obtain elements as the result of relationships with other entities.
- An element may be defined only once to a model.
- If an element exists in more than one entity, it must do so by means of a relationship.

For example, the Customer Code is an element of the Customer entity. The Order entity might also have a Customer Code element. Since Customer Code can be defined only once, the Customer Code in the Order entity must be the result of a join relationship with the Customer entity.

In the LANSA database files that are created, the names of the Customer Code fields will be based on the definition of the relationship between the two entities. For details about the field naming conventions for inherited elements, refer to 7.5 Relationships and Elements.

If you are not familiar with the rules of data modeling and how elements and relationships work together, you should attend the LANSA Modeling and Design Workshop. This guide does not teach the principles of data modeling.

6.3 Elements and LANSA

Elements are converted to fields in the LANSA Repository.

When defining an element, its physical details are either derived from its data type, or from type and length details provided by the user.

In the case of an element belonging to an External Entity, its physical details are derived from its existing repository definition and cannot be changed.

If a data type has been specified for the element, it will have a reference field based on its data type. For example, the Customer Name field will have a reference field of STD_NAME if based on the Name data type.

For more information about data types, refer to 4. Data Types.

6.4 Working with Elements

Elements can be created as unattached elements or as part of a specific entity. Unattached elements can be attached to entities at a later stage and vice versa. Refer to 5.6.6 Transferring Elements for information about attaching and detaching elements.

The Element dialogs allow you to:

- 6.4.1 Create an Element
- 6.4.2 Change an Element
- 6.4.3 Delete an Element
- 6.4.4 Maintain Element Notes.

6.4.1 Create an Element

To create an element, click the 🚨 button on the toolbar in the Unattached Elements or Entity Elements window. The Create Element dialog is displayed:

Create Element					
Element name					
Definition					
Data type	Amount				•
○ Type/length	Packed	¥	Length 9	<u>~</u>	Dec
Field name]			
			OK		Cancel

An element name is mandatory. It can be up to 20 characters in length, of mixed case and must be unique within the model.

Either a data type or a field type and length is required. To specify a data type, select the Data type option and select an appropriate type from the drop-down list. For more information about data types, refer to 4. Data Types.

To specify a field type and length, check the corresponding radio button, select a type from the drop-down list and enter a length and number of decimal places if appropriate.

A field name is optional. It will be the name of the field in the LANSA Repository when the element is built. It can be up to 9 characters in length, should follow normal field naming standards and must be unique within the model.

Note that, at this point, the entry of a field name is optional – it is not required for the purposes of building the data model. However, it must be specified when building the entity to which it is attached. You can specify the field name using the 6.4.2 Change an Element dialog. If a field name is not specified when building the element, a build error for the element will be generated.

Press the *Cancel* button if you do not wish to create the element.

Press the *OK* button to create the element. Once it has been validated, the new element will be added to the Unattached Elements or Entity Elements list as applicable. The Create Element dialog will be reset to allow you to enter more element details. Press *Cancel* when you have finished creating elements.

6.4.2 Change an Element

To change an element's definition, open the Unattached Elements list or Entity Elements list and:

- double-click the entry in the list that you wish to change, or
- select the element in the list and click the 💷 button on the toolbar.

If the element to be changed is not inherited by the entity by means of a relationship, the following dialog is displayed:

Change Element	
Element name Definition	Customer Name
🔿 Data type	Amount
Type/length	Alphanumeric Length Dec 25 0
Field name	LPCCUSTNM
	OK Cancel

Any part of the element's definition may be changed.

The element name is mandatory. It can be up to 20 characters in length of mixed case and must be unique within the model.

A data type or a physical field type and length is required.

To specify a Data type, select the *Data type* option and select an appropriate type from the drop-down list. Note that the data type's type and length is displayed in the Type and Length controls below the data type dropdown list. For more information about data types, refer to 4. Data Types.

To specify a Field Type and Length, select the *Field* radio button, select a type from the drop-down list and enter a length and number of decimal places as appropriate.

A field name of up to nine characters is optional. The field name will be the name of the field in the LANSA Repository when the element is built. It can be up to nine characters in length, should follow normal field naming standards and must be unique within the model.

Note that, at this point, the entry of a field name is optional – it is not required for the purposes of building the data model. However, it must be specified when building the entity to which the element is attached. If a field name is not

specified when building the element, a build error for the element will be generated.

Press the *Cancel* button if you do not wish to change the element.

Press the *OK* button to change the element. Once it has been validated, the Unattached Elements or Entity Elements list will be updated with the element's new details.

Inherited Elements

If the element to be changed has been inherited by the entity by means of a relationship, the following dialog will be displayed:

📩 Maintain Inherit	ed Element 🛛 🗙
Element name	Customer Number
From entity	Customer
Field name	LPCCUSNUM
Reset	OK Cancel

Only the field name of the inherited element can be changed. The field name will be the name of the field in the LANSA Repository when the element is built. It can be up to nine characters in length, should follow normal field naming standards and must be unique within the model.

By default, the field name of the element will be the same as that of the element in the entity from which it has been inherited. Note that, if the same element has been inherited more than once by means of multiple relationships, the inherited elements must have unique names within the inheriting entity. Refer to 7.7.4 Creating Multiple Relationships.

Press the *Cancel* button if you do not wish to change the element.

Press the Reset button to reset the inherited element's field name back to its original field name.

Press the *OK* button to change the element. Once it has been validated, the Entity Elements list will be updated with the element's new details.

6.4.3 Delete an Element

To delete an element from the Unattached Elements or Entity Elements list, select it and click the 💌 button on the toolbar.

Note that elements cannot be deleted form an External Entity.

You will be asked to confirm the element's deletion. Select *Yes* to delete the element, or the *No* to *Cancel* the deletion.

Note that he deletion behavior depends on the element list from which you are deleting:

- If you are deleting the element from the *Entity Elements* list, it is detached and becomes available in the *Unattached Elements* window for later use if required.
- If deleted from the *Unattached Elements* window, the element is completely deleted from the model.
- If an element that has been inherited via a relationship is deleted from the Entity Elements list, the entire corresponding relationship with the opposing entity is also deleted.

Note that deleting an element will not delete the corresponding field from the LANSA Repository.

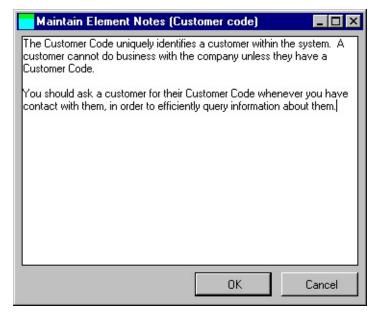
6.4.4 Maintain Element Notes

Element notes are transformed into field help text when the element is built in the LANSA Repository.

Note that notes cannot be maintained for elements belonging to an External Entity.

To maintain an element's notes, select the element in the Unattached Elements

or Entity Elements list and click the 📴 button on the toolbar. The Maintain Element Notes dialog is displayed:



The standard LANSA special help text characters may be used in the notes. For example, you can display help text underlined, highlighted, etc.

If the notes being maintained are that of an inherited element, the original notes may be overwritten for the inherited element, thus providing contextualized notes according to the use of the inherited element. In addition, a Reset button is available to set the element's notes back to that of the original notes.

Once the notes have been maintained, press the *OK* button to update them. If you do not wish the notes to be updated, press the *Cancel* button.

A notepad icon appears to the right of the element name in the Unattached Elements and Entity Elements lists if the element has notes.

Note that contextualized notes for an element inherited by means of a relationship may only be maintained if a unique field name for the inherited element has been specified. If a unique field name has not been specified, the

original element notes will appear in the dialog to be maintained.

7. Relationships

If you have data modeling experience, the concept of relationships will be very familiar to you. If you are unfamiliar with data modeling, you should attend the LANSA Modeling and Design Workshop as this guide is not designed to teach you the principles of data modeling.

Once you have created your entities, you can define relationships between them. The Logical Modeler supports extended Chen E-R relationships. These include the basic Parent/Child and Join relationships along with extensions to Chen E-R relationships such as Optional Join, Variation and Include.

Relationships are very important. At a data level, a Parent/Child relationship will cause the keys of a parent entity to cascade to the child entity.

Relationships influence the file keys, logical files, referential integrity validation rules and access routes which are generated when the model is built in the LANSA Repository.

This information is described in:

- 7.1 Relationship Types
- 7.2 Relationship Rules
- 7.3 Source and Target Entities
- 7.4 Relationships and Entities
- 7.5 Relationships and Elements
- 7.6 Relationships and LANSA
- 7.7 Work with Relationships

7.1 Relationship Types

Following is a brief summary of the extended Chen E-R Relationships.

- Parent/Child
- Join
- Include
- Variation

Note that the Abstraction relationship is not implemented in the Logical Modeler. However, Abstractions can be created to group objects together. Refer to 8. Abstractions.

Parent/Child

One entity is said to be the Parent of another if the data of the Child entity belongs to (is contained within) the Parent. Parent/Child relationships can be identified by asking:

- Is the entity's identifier unique?
- Can the entity exist on its own?

If the related entity's identifying element is unique, that is, you can identify an occurrence in the entity solely by its identifying element, then the entity probably has no parents. If you require other information to identify an occurrence, that is, another entity must exist before it can exist, then the entity probably has a parent.

For example, an Ordered Product cannot exist on its own. The Ordered Product must have a parent, Order, before it can exist.

The identifying elements of the Parent entity are inherited by the Child entity. These elements are often called Parent Keys or Foreign Keys.

Also see Parent/Child Rules.

Join

One entity is said to be joined (or to refer) to another if a data (that is, non-key) element or group of elements from the first entity can be used to access an occurrence of the second.

The Join relationship differs from the Parent/Child relationship in that the element(s) used to access the referenced entity are not necessary to identify the primary (or referencing) entity. However, they are still inherited to become Foreign Keys.

For example, an Order is placed by a Customer. The Order is identified by its Order Number. It will look up a Customer Code in the Customer entity. The Join relationship is further classified as a Mandatory Join or an Optional Join. The Mandatory Join is described as a must refer to relationship. The Optional Join is described as a may refer to relationship.

Also see Join Rules.

Include

The Include relationship allows elements from one entity to exist in many different entities. However, unlike a Parent/Child and Join relationships, no foreign keys result. The elements become part of the entity.

For example, a conceptual entity called Address is created with attributes of Street, State, City, Country and Post Code. This entity has no identifying element. The Address entity could be included into a Customer entity or Supplier entity. The Address elements will become part of the Customer file and Supplier file instead of being in a separate Address file.

Also see Include Rules.

Variation

This type of relationship is used in situations where elements within an entity are dependent on the value of another element. For example, a Product can either be manufactured or sourced from a third party supplier. If the product is manufactured, a drawing number may be required. If the product is sourced from a supplier, a supplier code is required. Rather than create a single Product entity with all elements for manufactured and supplied products, three entities are created. Product will have variations called Manufactured Product and Supplied Product. Storing all the elements in a single Product entity would result in a sparsely populated database, i.e. many fields will be unused.

Once a Product has been created and designated as Manufactured, the same product cannot be designated as Supplied.

Note that the database, once built, will have no awareness of how variations of the original entity are accessed. An element should be created in the original entity to hold a value which can be used in determining which database file variant information should be retrieved from in your application.

Also see Variation Rules.

7.2 Relationship Rules

Allowable relationships are defined based on entity type as follows:

Relationship Type	Source Entity Type	Target Entity Type
Parent/Child	Data, Variant or External	Data
Join	Data or Variant	Data, Variant or External
Variation	Data, Variant or External	Variant
Include	Data or Variant	Conceptual

For more information about how to define Source and Target entities, refer to 7.3 Source and Target Entities and 7.7.1 Create a Relationship.

Following is a summary of the Logical Modeler Relationships Rules for:

- Parent/Child Rules
- Join Rules
- Include Rules
- Variation Rules

Note that Abstractions are not entities. They are objects used to group entities. Abstractions cannot be used in relationships.

Parent/Child Rules

Parent child relationships may be created between data, variant and external entities.

The identifying elements of the Parent entity are inherited by the Child entity. These elements are often called Parent Keys or Foreign Keys.

Any of the inherited identifying elements that would be duplicated as a result of multiple Parent/Child relationships will be shared automatically by the entity.

You are not allowed to create a loop structure using parent/child relationships. For example, if entity A is the parent of entity B, then entity B cannot be the parent of entity A. Or, if entity A is the parent of entity B and entity B is the parent of entity C, then entity C cannot be the parent of entity A or B.

In addition, an External Entity can only be the Parent entity in a Parent/Child

relationship.

Join Rules

Two entities are allowed to have multiple join relationships. It is recommended that you use the descriptive prefix to identify the relationships.

Any of the inherited identifying elements that would be duplicated as a result of multiple relationships can be shared by using the *Share inherited identifiers* option.

A Variation relationship must exist for a Variant entity before you can create Join relationships using that Variant entity. Once the Variation relationship is created, the identifying elements are known in the Variant entity.

Note that an External Entity can only be the Target entity in a Join relationship.

Include Rules

Two entities are allowed to have multiple include relationships. It is recommended that you use the descriptive prefix to identify the relationships.

The target entity for an include relationship must be a conceptual entity.

You cannot create include relationships between conceptual entities.

An Includes relationship cannot be associated with an External Entity.

Variation Rules

The source entity for a variation relationship can be a data, variant or external entity.

The target entity for a variation relationship must be a variant entity.

You may create only one variation relationship to a variation entity.

A Variation relationship must exist for a Variant entity before you can create Join relationships using that Variant entity. Once the Variation relationship is created, the identifying elements are known in the Variant entity.

7.3 Source and Target Entities

Relationships are established between two entities. The types of entities determine the type of relationships supported. In addition, the entities must be identified as source and target:

- The source entity is the entity which is initiating the relationship.
- The target entity is the object which receives the relationship.

The concept of source and target are determined when the relate connector is linked to the entity. Refer to 7.7.1 Create a Relationship.

For example, Product has a Variation relationship to Manufactured Product. Product is a Data entity and must be the source. Manufactured Product must be a Variant entity and must be the target. The source end of the Relate connector must attached to the Product entity and the target end of the Relate connector must be attached to the Manufactured Product.

7.4 Relationships and Entities

Relationship rules are defined based on entity type. Remember, entities are defined before relationships. The Logical Modeler only allows meaningful relationships to be defined for a given entity type as the logical model can be built into a physical database. For example, a Parent/Child relationship can be defined only between Data, Variant and External entities.

Relationship TypeSource Entity TypeTarget Entity TypeParent/ChildData, Variant or ExternalDataJoinData or VariantData, Variant or ExternalVariationDataVariant or ExternalIncludeDataConceptual

These are the default relationship rules:

For more information about how to define Source and Target entities, refer to 7.3 Source and Target Entities and 7.7.1 Create a Relationship.

Also see 7.2 Relationship Rules.

Note: Abstractions are not entities. They are objects used to group entities. Abstraction cannot be used in relationships.

7.5 Relationships and Elements

In order for Parent/Child or Join relationships to be properly built, both source and target entities should have an identifying element. Identifying elements are not required when the relationship is first defined. However, an error will result when the entity is built if no identifying element has been defined.

For variation relationships, the source entity must be a data or external entity and must have an identifying element. The target entity must be a variant entity and must not have an identifying element.

For include relationships, the source entity must be a data entity. The target entity must be a conceptual entity and no identifying element is allowed for the conceptual entity.

For a variant entity, a Variation relationship must exist before you can create Join relationships using that variant entity as the target entity. Once the Variation relationship is created, the identifying elements are known in the variant entity.

The elements of an entity are directly impacted by the relationships defined. For more details, refer to 6.2 Elements and Relationships.

7.6 Relationships and LANSA

Relationships play an important role in defining the database built in the LANSA Repository. Relationships are responsible for creating the following physical aspect of a database file:

- access routes
- logical views
- referential integrity validation rules.

LANSA access routes are used by templates and provide important navigation information about the database. The access routes created in LANSA are the direct result of relationships.

Logical views are based on the second & subsequent Parent/Child relationships and the Join relationships between entities. When an Order refers to a Customer, the Order entity will have an additional logical view based on the Customer Code to provide access from Customer to Order.

File validation rules for referential integrity are the direct result of relationships. A validation check that prevents an Order from being deleted when Ordered Products still exist is the result of the Parent/Child relationship between Order and Ordered Product.

7.7 Work with Relationships

You will work with relationships using 2.4.4 The Document Stencil and by 2.4.6 Performing Actions on Diagram Objects.

The following dialogs allow you to:

- 7.7.1 Create a Relationship
- 7.7.2 Change a Relationship
- 7.7.3 Delete a Relationship

If you are creating more than one relationship between two entities, you should be familiar with 7.7.4 Creating Multiple Relationships.

7.7.1 Create a Relationship

To create a relationship between any two entities, drag-and-drop the Relate connector shape from the Document Stencil onto a blank portion of the model diagram.

Each end of the Relate connector must be attached to an entity. To attach the connector to an entity, drag one end of the connector to the center of the entity that is the source entity in the relationship. Drop the end of the connector when the border of the source entity turns red. If the border of the entity is not red, it will not be attached correctly to the connector. Once you have connected the source entity, drag-and-drop the other end of the connector to the target entity in the same manner.

The source end of the connector is the left-hand end of the connector. The target end of the connector is the right-hand end of the connector.

If you attempt to create an invalid relationship, an error message will be displayed and the connector will be deleted. For example, if you attempt to use an Abstraction in a relationship, an error will occur.

If both ends of the connector have been correctly attached to the entities, the Create Relationship dialog is displayed:

Create Relationship
Туре
Order
is a parent of
Customer
Share inherited identifiers
Descriptors
Order
is a parent of
Customer
Customer
is a child of
Order
Descriptive prefix
Customer (TBCUST) Physical Object
Logical view name
OK Cancel

Alternatively, clicking the 🗗 button on the main toolbar or selecting the Specify Relationship... option from the Edit pull-down menu will enable you to create a relationship using the following dialog, which is similar to that above:

Create Relationship
Relationship
Customer 🗾
includes 🗾
Address 🗾
Share inherited identifiers
Descriptors
Customer
Address
Address
Customer
Descriptive prefix
Customer Physical Object
Logical view name
OK Cancel

This dialog differs from the one displayed when using the drag-and-drop method in that it lists all available entities in two drop-down lists either side of the relationship type drop-down list. Select the two entities to be related from the drop-down lists.

For either method, the type of relationship being defined between the two entities must be selected. The Type drop-down list will only contain the relationship types that are valid between the two entities. The following table shows which relationships are valid between which entity types:

Source Entity Type	Target Entity Type	Valid Relationships
Data	Data	Parent/Child
		Mandatory Join
		Optional Join
Data	Conceptual	Includes
Data	Variant (with no existing Variation	Variation

relationship)

Data	Variant (with existing Variation relationship)	Mandatory Join Optional Join
Data	External	Mandatory Join Optional Join
Variant	Data	Parent/Child Mandatory Join Optional Join
Variant	Conceptual	Includes
Variant	Variant	Variation
Variant	External	Mandatory Join Optional Join
External	Data	Parent/Child
External	Variant	Variation

The Share inherited identifiers is used to prevent duplication of the inherited identifying elements. When a relationship is being defined and more than one relationship already exists, then the identifying elements of the source entity can be shared with the target entity. Refer to 1.1.10 Understanding Shared Keys.

Source and Target Descriptors are optionally used to describe the relationship between the two entities from both directions. They are defaulted according to the type of relationship being created, but can be blanked out or overridden with more meaningful descriptions. They serve two functions:

- On the model diagram, the resultant relationship connector, once created, will contain the Source-to-Target description entered here, followed by the Computer Prefix in brackets if it has been specified.
- Descriptions of logical views, access routes and referential integrity validation rules created in the repository will use the text specified here.

A *Descriptive prefix* is optional. If specified, it is appended to the start of the element names which are inherited as a result of the relationship. Using a

Descriptive prefix helps make the element and resultant field visually unique, as its description is a combination of the computer prefix and the name of the inherited element(s).

Consider this example of Descriptive prefix use:

- The Customer entity has two include relationships with the Address entity: one for delivery address, and one for billing address.
- The descriptive prefix for the first relationship is specified as Delivery. If the Address entity has an element called Zip Code, the field generated in the repository will have a description of Delivery Zip Code.
- The descriptive prefix for the second relationship is specified as Billing. If the Address entity has an element called Zip Code, the field generated in the repository will have a description of Billing Zip Code.

A Logical View Name may be specified if the relationship being created is a join relationship. Note that, if a physical file name of greater than the maximum number of characters allowed (10 if the *RPGIV setting is enabled in your system, 8 if it is not) has been specified for the source entity, a logical view name must be specified before the source entity will successfully build. If, however, a physical file name has been specified that is less than the maximum allowed by two or more characters has been specified, the logical view name is optional. In this instance, the modeler will automatically create a logical view name when the entity is built. This view name will be based on the source entity's physical file and a sequential two digit number.

Note that if an entity inherits the same element multiple times (as with the example above), the field names for the inherited elements must be unique within the inheriting entity. Refer to 6.4.2 Change an Element.

Press the *OK* button to create the relationship. Once validation has been passed, the elements inherited by the source or target entity will be shown in the Elements view of the model diagram. In addition, the build status of the entities affected by the relationship will be updated.

Press the *Cancel* button if you do not wish to create the relationship. The Relate connector shape you placed on the diagram will be deleted.

If you are creating more than one relationship between two entities, you should be familiar with 7.7.4 Creating Multiple Relationships.

Reminder: You are not allowed to create a loop structure using parent/child relationships. For example, if entity A is the parent of entity B, then entity B cannot be the parent of entity A. Or, if entity A is the parent of entity B and

entity B is the parent of entity C, then entity C cannot be the parent of entity A or B.

7.7.2 Change a Relationship

To change a relationship's details, select its shape on the diagram and:

- select the *Change*... command from its right-click menu, or
- select the *Change*... command from the Edit menu, or
- click the 🔟 button on the toolbar.

The Change Relationship dialog is displayed:

Change Relationship	
Туре	
Order	
must refer to	
Customer	
Share inherited identifiers	
Descriptors	
Order	
must refer to	
Customer	
Customer	
must be referred to by	
Order	
Descriptive prefix	
Order (TBORD) Physical Object	
Logical view name	
OK	Cancel

The type of the relationship may not be changed. If the type of relationship is incorrect between the two entities, it should be deleted and an appropriate relationship created.

If applicable, the sharing of identifying elements with the target entity can be enabled or disabled by checking or unchecking Share inherited identifiers. Refer to 1.1.10 Understanding Shared Keys.

Source and Target Descriptors are optionally used to describe the relationship between the two entities from both directions. They are defaulted according to the type of relationship being created, but can be blanked out or overridden with more meaningful descriptions. They serve two functions:

- On the model diagram, the relationship connector, once created, will contain the Source to Target description entered here, followed by the Computer Prefix in brackets if one has been specified.
- Descriptions of logical views, access routes and referential integrity validation rules created in the repository will use the text specified here.

A computer prefix is optional at all times. If specified, it is appended to the start of the element names which are inherited as a result of the relationship. Using a Descriptive prefix helps in making the element and resultant field visually unique, as its description is a combination of the computer prefix and the name of the inherited element(s).

Consider this example of descriptive prefix use:

- The Customer entity has two include relationships with the Address entity: one for delivery address, and one for billing address.
- The descriptive prefix for the first relationship is specified as Delivery. If the Address entity has an element called Zip Code, the field generated in the repository will have a description of Delivery Zip Code.
- The descriptive prefix for the second relationship is specified as Billing. If the Address entity has an element called Zip Code, the field generated in the repository will have a description of Billing Zip Code.

A Logical View Name may be specified if the relationship being changed is a join relationship, or if the relationship is a second or subsequent parent/child relationship. Note that only parent/child relationships subsequent to the first will generate logical views. If the sequence of multiple parent/child relationships is changed within the affected entity, logical view names that have been previously specified may be reset, because the first parent/child relationship will not generate a logical view. If this reset occurs, the logical view name(s) of parent/child relationships subsequent to the first must be respecified.

Note that, if a physical file name of greater than the maximum number of characters allowed (10 if the *RPGIV setting is enabled in your system, 8 if it is not) has been specified for the source entity, a logical view name must be specified before the source entity will successfully build. If, however, a physical file name has been specified that is less than the maximum allowed by two or more characters has been specified, the logical view name is optional. In this instance, the modeler will automatically create a logical view name when the entity is built. This view name will be based on the source entity's physical

file and a sequential two digit number.

Note also that if an entity inherits the same element multiple times (as with the example above), the field names for the inherited elements must be unique within the inheriting entity. Refer to 6.4.2 Change an Element.

Press the *OK* button to change the relationship. Once validation has been completed, the build status of the entities affected by the relationship will be updated and the relationship shape will be updated appropriately on the diagram.

Press the *Cancel* button if you do not wish to change the relationship.

7.7.3 Delete a Relationship

To delete a relationship, select its shape on the diagram and:

- select the *Delete*... command from its right-click menu, or
- select the *Delete*... command from the Edit menu, or
- click the ຶ button on the toolbar.

You will be asked to confirm the deletion.

Upon deletion, the build status of any affected entities will be reset appropriately and any inherited elements will be removed from the Elements view of the diagram.

Note that a relationship may also be deleted by deleting an element (via the Entity Elements dialog) which has been inherited by an entity as a result of the relationship.

7.7.4 Creating Multiple Relationships

When a relationship is created between two entities, inherited element(s) will be added to one of the entities. For example, if the Order entity has a refers to relationship with the Customer Entity, then the identify attribute of Customer Number becomes an inherited element in the Order entity. In this example, the Order entity has only one element named Customer Number and you may successfully build this model.

When more than one relationship is created between to entities, then the inherited element will appear more than once. For example, the Customer entity might include the Address conceptual entity twice as follows:

Customer includes a billing Address.

Customer includes a shipping Address.

A descriptive prefix such as "billing" and "shipping" can be used to name the inherited elements from the two different relationships, but the underlying field names will be the same. To successfully build this model, you must have unique field names for all elements in the entity. Hence, you must change the field names of inherited elements in the file to ensure that they are unique. For example, you might add a suffix such as SHP to the field names of the inherited elements used for the shipping address. An element with a field name of POSTCD could be renamed POSTCDSHP.

Refer to Inherited Elements in the 6.4.2 Change an Element.

8. Abstractions

An abstraction is like a container object and is used to a group entities. It is very useful when working with large data models. For example, an abstraction called Order Processing can be used to group the Customer, Product, Order and Ordered Product entities. By activating an abstraction, you can now work with just the Order Processing part of the model (four entities) rather than all entities in the model. Entities which are not part of the abstraction are not displayed in the diagram.

Abstractions are not entities and cannot be used in relationships.

Refer to 8.3 Work with Abstractions.

8.1 Abstractions and the Model

A model can contain any number of abstractions. As well as an aid to simplify the diagram view of the model, they can be used to restrict the model's build activity to those entities contained within the abstraction.

Abstractions are not transformed into repository objects when the model is built . They exist only to assist you when constructing the model.

8.2 Abstractions and Other Objects

Abstractions can contain entities and other abstractions. When an abstraction is activated, only those entities and abstractions contained within the abstraction will be shown on the model diagram. Only those relationships that link the entities within the abstraction are shown. Refer to 8.3.11 Hiding Complexity. In addition, the Build Model function is restricted to those entities within the

abstraction, enabling you to build discreet parts of the model. Changes to entities and relationships within the abstraction are still propagated to the entire model, however, and can affect entities that are not included in the abstraction.

8.3 Work with Abstractions

You will work with abstractions using 2.4.4 The Document Stencil and by 2.4.6 Performing Actions on Diagram Objects.

When working with abstractions, you should be familiar with the following:

- 8.3.1 Create an Abstraction
- 8.3.2 Change an Abstraction
- 8.3.3 Delete an Abstraction
- 8.3.4 Specify Abstraction Objects
- 8.3.5 Another Way to Add Objects to an Abstraction
- 8.3.6 Another Way to Remove Entities from an Abstraction
- 8.3.7 Activate an Abstraction
- 8.3.8 Deactivate an Abstraction
- 8.3.9 Abstraction Notes
- 8.3.10 Building the Model while an Abstraction is Active
- 8.3.11 Hiding Complexity.

8.3.1 Create an Abstraction

To create an abstraction, drag-and-drop the Abstraction shape from the document stencil onto the model diagram. The Create Abstraction dialog is displayed:

Create Abstraction			
Name	[
		OK	Cancel

A name for the abstraction must be specified. It can be up to 20 characters in length of mixed case, and must be unique within the model.

Click the *OK* button to create the abstraction. Its shape will be updated with its name on the model diagram.

Note: If creating an abstraction when another abstraction is active, the new abstraction will be automatically added to the active abstraction.

If you do not wish to create an abstraction, click the *Cancel* button.

8.3.2 Change an Abstraction

To change an abstraction's name, select the abstraction shape in the diagram and:

- select the *Change*... command from its right-click menu, or
- select the *Change*... command from the Edit menu, or
- click the 🔟 button on the toolbar.

The Change Abstraction dialog will be opened.

Change the Abstraction's name as required. It can be up to 20 characters in length of mixed case, and must be unique within the model.

Click the *OK* button to change the abstraction. Its shape will be updated with its new name on the model diagram.

If you do not wish to change the abstraction, press the *Cancel* button.

8.3.3 Delete an Abstraction

To delete an abstraction, select the abstraction shape to be deleted in the diagram and:

- select the *Delete*... command from its right-click menu, or
- select the *Delete*... command from the Edit menu, or
- click the 📕 button on the toolbar.

You will be asked to confirm the deletion. Select *No* if you do not wish to delete the abstraction, otherwise select *Yes* to delete the abstraction and remove it from the model diagram. Any abstractions to which it belongs will be updated accordingly.

Note that an abstraction cannot be deleted if it is currently active. In order to delete it, it must first be deactivated.

8.3.4 Specify Abstraction Objects

To specify the objects that are to be contained within an abstraction, select its shape and:

- select the Specify Objects... option from its right-click menu, or
- select the *Specify Objects*... option from the Abstraction menu.

The Specify Abstraction Objects window is displayed:

Specify Abstraction Objects (C	Irder Processing) 📃 🗖 🗙
Object	Abstraction
Customer	
Order	
Address	
Crdered Product	
Product	
Manufactured Product	
Supplied Product	
Supplier	
Warehouse	
Product Stock	
	OK Cancel

To add objects to the abstraction, double-click their entries in the left-hand Object list or drag-and-drop them into the right-hand Abstraction list. As objects are added to the Abstraction list, they are removed from the Object list. Alternatively, you can use 8.3.5 Another Way to Add Objects to an Abstraction.

To remove objects from an abstraction, double-click their entries in the Abstraction list or drag-and-drop them into the Object list. As objects are removed from the Abstraction list, they are added to the Object list. Alternatively, you can use 8.3.6 Another Way to Remove Entities from an Abstraction.

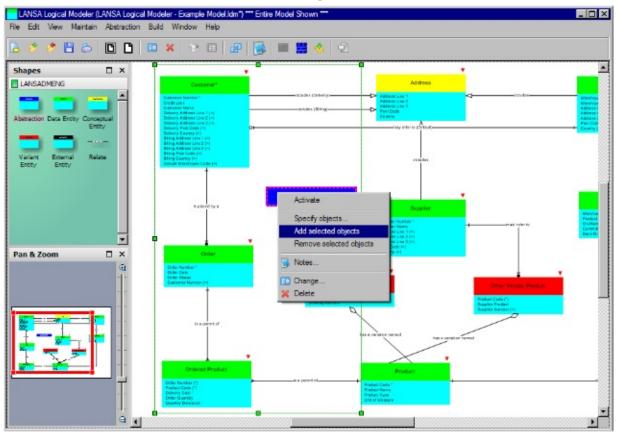
Click *OK* to update the abstraction. The Abstraction's diagram shape will be updated appropriately.

Click the *Cancel* button if you do not wish to specify objects for the abstraction.

8.3.5 Another Way to Add Objects to an Abstraction

As well as being able to specify objects for an abstraction as described in 8.3.4 Specify Abstraction Objects, you can do so by interacting with the diagram alone.

In the following example, the Customer, Order and Ordered Product entities are about to be added to the Order Processing abstraction.



To add objects to an abstraction:

- 1. Select its shape on the model diagram.
- 2. Hold down the shift key and select all entities and abstractions that you wish to add to the abstraction.
- 3. Once you have selected all objects to be added:
- Select the *Add Objects*... command from the abstraction's right-click menu, or
- select the *Add Objects...* command from the Abstraction menu.

The abstraction shape will be updated with the name of the entities and abstractions you have added to it.

Note that any entities or abstractions created are automatically added to the active abstraction. Refer to 8.3.7 Activate an Abstraction for how to activate an abstraction.

Note also that this method of adding objects to an abstraction cannot be used if the abstraction is currently active. Instead, use the Specify Objects... command from the abstraction's context menu (i.e. right-click with the mouse).

8.3.6 Another Way to Remove Entities from an Abstraction

To remove entities or other abstractions from an abstraction, its shape should first be selected on the model diagram.

Holding down the shift key, select all entities and abstractions that you wish to remove from the abstraction.

Once you have selected all objects to be removed:

- select the *Remove Objects*... command from the abstraction's right-click menu, or
- select the *Remove Objects*... command from the Abstraction menu.

The abstraction shape will be updated accordingly.

8.3.7 Activate an Abstraction

To activate an abstraction:

- select the Activate... command from its right-click menu, or
- select the Abstraction shape in the diagram and select the *Activate*... command from the Abstraction menu.

The model diagram will be re-drawn to show only the abstraction and the objects within it. The title bar of the modeler will be changed to indicate that the abstraction is active and being viewed.

Important Notes:

- Only relationships between entities within the abstraction are shown, but entities outside the abstraction can be affected by changes to those within it.
- It is possible to drill down into abstractions if the abstraction themselves contain other abstractions.
- The relative positions of the entities in the abstraction may be changed once the abstraction is activated. Entities will assume these positions whenever the abstraction is activated.
- The scope of the Build Model function is limited to those entities within the active abstraction.

8.3.8 Deactivate an Abstraction

To deactivate an abstraction:

- select the *Deactivate*... command from its right-click menu, or
- select the Abstraction shape in the diagram and select the *Deactivate...* command from the Abstraction menu.

The model diagram will be re-drawn appropriately, with entities returning to their original positions if they were moved whilst the abstraction was active. The title bar of the modeler will be changed to indicate whether an abstraction or the entire model is being viewed.

If the abstraction is part of a drill-down from another abstraction, the previous abstraction will become active.

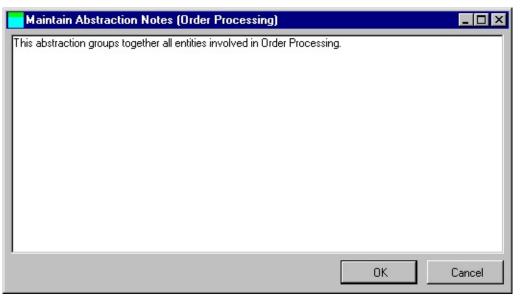
8.3.9 Abstraction Notes

The notes facility for abstraction exists merely as a documentation tool for use when constructing a data model. Unlike element notes, which are transformed into field help text when the element is built, abstraction notes are not carried through from the model to the repository.

To maintain an abstraction's notes, select the abstraction's shape on the diagram and:

- select the *Notes*... option from the abstraction's right-click menu, or
- select the *Notes*... option from the Edit menu, or
- click the 🗾 button on the toolbar.

The Maintain Entity Notes dialog is displayed:



The standard LANSA special help text characters may be used in the notes. For example, you can display help text underlined, highlighted, etc.

Once the notes have been maintained, press the *OK* button to update them. If you do not wish the notes to be updated, press the *Cancel* button.

On the model diagram, an asterisk (*) will appear after the abstraction name if it has notes attached.

8.3.10 Building the Model while an Abstraction is Active

It should be noted that using the Build Model function whilst an abstraction is active will cause the modeler to only analyze those objects contained within the abstraction and not the entire model.

8.3.11 Hiding Complexity

Abstractions can contain entities and other abstractions. When you include an abstraction object within another abstraction, you create a drill-down effect.

When an abstraction is activated, only those entities and abstractions contained within the active abstraction will be shown on the model diagram. Only those relationships that link the entities within the abstraction are shown. You can create a new abstraction with the active abstraction or you may activate another abstraction if it is listed as part of the active abstraction.

If you activate another abstraction within the active abstraction, only the objects in the newly activated abstraction will be shown. You are able to drill-down to the next level of the abstraction.

Using the drill-down structure, abstractions can be used to hide the complexity of the model. For example, you may hundreds of entities in a model diagram. You can create a new abstraction named "My Model" with nothing in it. When the My Model abstraction is activated, all objects in the model will disappear. The diagram will only show the My Model Abstraction. In effective, you have hidden the complexity of the diagram. You now have your own abstraction in which you can create your own small model.

If you wish to organize your entities into smaller models, you can also use the drill-down structure to create an abstraction menu to access these smaller models. For example, you may have an abstraction named Order Processing with all entities used for your Order Processing database. You may have an abstraction named Inventory Management with all entities used for your Inventory Management database. Simply create a new abstraction named "My Models" that contains the Order Processing and Inventory Management abstraction. When you activate "My Models", you will only see the Order Processing and Inventory Management abstractions. In effect, this is a menu to your other models. You can now activate the model you wish to develop.

It is very important that you do not create a loop in your abstractions when drilling down. For example, if Abstraction A contains Abstraction B, and Abstraction B contains Abstraction C, then do not include Abstractions A or B into C.

9. User Views

User Views are not a logical modeling concept. They are included to allow analysts to document logical views of the data which can be used by the physical database.

The aim of a user view is to provide access to a file in a sequence other than the primary key and logical views created as a result of relationships. User views will generate logical views when a database build is performed on an entity.

For example, a logical view sequenced by Customer Name will not ordinarily be created for the Customer entity, as there are no relationships involving Customer Name and it is not part of the key of Customer. However, an end user may wish to view Customer information in Customer Name sequence. A user view can be created to support this requirement.

User views can only be specified for entities which become physical files when they are built. For example, Data and Variant objects may have user views. User views become logical files in the repository when the entity is built. You may add user views as a means of creating logical files which may be needed for reporting or other operations.

This is described in:

- 9.1 User Views and Entities
- 9.2 User Views and Elements
- 9.3 User Views and LANSA
- 9.4 Maintain User Views.

9.1 User Views and Entities

A user view can only be defined for an entity that is built into a physical file. User views can be specified for data and variant entities.

User views cannot be specified for conceptual or external entities.

For example, the Customer data entity can have user view defined but the Address conceptual entity cannot.

9.2 User Views and Elements

A user view can use any element in an entity, including those that are inherited as the result of relationships with other entities.

For example, join or parent keys can be used in a user view.

9.3 User Views and LANSA

A user view becomes a logical file in LANSA when the entity is built. Again, only entities which become physical files may have user views.

9.4 Maintain User Views

To maintain user views for an entity, select its shape and:

- select the User Views... option from the entity's right-click menu, or
- select the User Views... option from the Edit menu, or
- click the 💷 button on the toolbar.

The Maintain User Views <name of entity> window is displayed:

Maintain User View	vs (Customer)	
🔌 🖪 🛛	×	
ibu quatamat name		
by customer name		

This example shows the user views for the Customer entity.

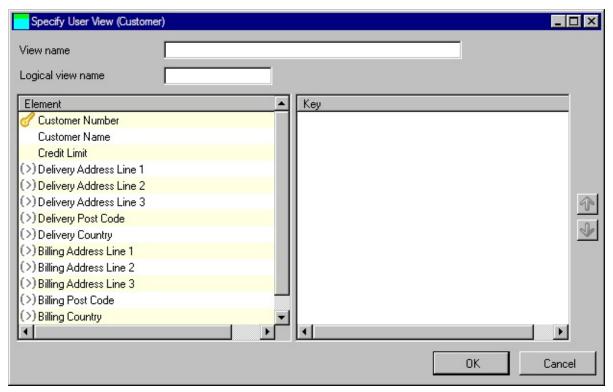
Using the toolbar buttons, from this window, you can:

- 9.4.1 Create a User View
- 9.4.2 Change a User View
- 9.4.3 Delete a User View.

Once you have finished maintaining user views, close the Maintain User Views window by clicking the subtron. The build status of the selected entity will be updated appropriately.

9.4.1 Create a User View

To create a user view, click the 🚨 button on the Specify User Views toolbar. The Specify User View dialog is displayed:



You must specify a Name for the view. It can be up to 40 characters of mixed case.

A Logical View Name should be specified if the physical file name of the affected entity is longer than the maximum number of characters allowed (10 if the *RPGIV setting is enabled in your system, 8 if it is not). If the physical file name of the affected entity is up to the maximum length allowed, no logical view name is required: the modeler will automatically generate a logical view name when the entity is built.

The left-hand side of the window shows a list of all the elements within the entity, including those that have been inherited as the result of relationship with other entities.

The right-hand side of the window will contain the key you select to control the sequence of the data displayed in the user view.

To specify the key for the view, double-click elements in the left-hand Elements list or drag-and-drop them from the left-hand Elements list onto the right-hand

Key list. As elements are added to the right-hand Key list, they are removed from the Elements list. Similarly, elements can be double-clicked or dragged and dropped from the Key list to the Elements list to remove them from the key. If the view has more than one element as the key, the key sequence can be changed by using the reasonable of the fright of the Key list. Press *OK* to create the view. After validation, the Maintain User Views window is redisplayed with the new user view shown in the list of views. Press *Cancel* if you do not wish to create the user view. The Maintain User Views window will be redisplayed.

9.4.2 Change a User View

To change a user view, double-click its entry in the Maintain User Views list or select it and click the 🔟 button on the toolbar.

The Specify User View window is displayed:

Specify User View (Custome	r)			_ 🗆 ×
View name	by customer name			
Logical view name				
Element		Key		
🕜 Customer Number		Customer Name		
Credit Limit				
(>) Delivery Address Line 1				
(>) Delivery Address Line 2				
(>) Delivery Address Line 3				
(>) Delivery Post Code				T
(>) Delivery Country				
(>) Billing Address Line 1				J.
(>) Billing Address Line 2				
(>) Billing Address Line 3				
(>) Billing Post Code				
(>) Billing Country				
(<) Default Warehouse Code				
•	•	•		Þ
			ОК	Cancel

You can change the Name of the view. It can be up to 40 characters of mixed case.

You can change the Logical View Name of the view. A Logical View Name should be specified if the physical file name of the affected entity is longer than the maximum number of characters allowed (10 if the *RPGIV setting is enabled in your system, 8 if it is not). If the physical file name of the affected entity is up to the maximum length allowed, no logical view name is required: the modeler will automatically generate a logical view name when the entity is built.

The left-hand side of the dialog shows a list of elements within the entity, including those that have been inherited as the result of relationship with other entities.

The right-hand side of the dialog contains the key of the user view.

Elements can be double-clicked or dragged and dropped between the two lists to

build the desired key.

As elements are added to the right-hand Key list, they are removed from the left-hand Elements list.

If the view has more than one element as the key, the key sequence can be changed by using the 🚳 and 🗣 buttons to the right of the Key list.

Press *OK* to change the view. After validation has been passed, the Maintain User Views window is redisplayed with the updated user view shown in the list of views.

Press *Cancel* if you do not wish to change the user view. The Maintain User Views window will be redisplayed.

9.4.3 Delete a User View

To delete a user view, select it in the Maintain User Views list and click the **X** toolbar button.

You will be asked to confirm the deletion. Select *No* if you do not wish to delete the view otherwise *Yes* to delete the view and remove it from the list.

Upon closing the Maintain User Views window, the entity's build status will be updated appropriately.

10. Building a Model

A build is the generation of a physical database in the LANSA Repository, based on your logical modeler data model. For example, a build of an entity will create repository fields, a physical file, logical files and an object access module.

The build is almost completely automated. Your database is built at the push of a button.

The following topics cover:

- How the logical modeler builds the database, with an illustration of the interdependencies between the different builds, the impact of changes and the need to rebuild a model. It explains what a build does and what you can expect as the result of the build.
- The fundamental steps and the rules for submitting and synchronizing the build of the prototype.
- What is involved and what is produced by each build. The LANSA items created by each build are explained.
- How to use the Builder Queue to manage your builds.

In the logical modeler, there are three steps in the building of a database and the steps in which they are run is described in 10.1 Sequencing the Builds. The order of the builds and the entities you select for building are very important because of entity dependencies. If you request a build which cannot be performed, the modeler will provide the appropriate error and warning messages. For example, a Relationship build expects both entities in a relationship to have a database build completed.

This information is described in:

- 10.1 Sequencing the Builds
- 10.2 Build Methods
- 10.3 Understanding Builds
- 10.4 Rebuilding Your Model
- 10.5 The Builder Queue
- 10.6 Build Tips and Techniques

10.1 Sequencing the Builds

The sequence of the builds and the entities you select for building are very important because of entity dependencies. In general, the sequence of the builds is as follows:

- Build Repository Fields
- Build Database File
- Build Relationships

Repository fields are the starting point for all work in LANSA. Fields must be defined before a file can be created. Files must be defined before relationships can be added.

It is important that you synchronize your building due to file relationships. An entity with no relationships to other entities could be built on its own. The Relationships build which needs to create a file look-up validation rule is expecting the other database file to exist.

For example, if you have three entities in your model, you should perform a Repository fields build for each of the three entities first. Once completed successfully, you would perform the database build for each entity. If all entity builds are successful, you would build the relationships.

The Logical Modeler provides a builder queue facility which simplifies the build process by synchronizing the entity builds for you.

Build Repository Fields

The repository fields build converts your entity's elements into LANSA Repository field definitions based on the element's data type or physical definition. The field definition is created and, if a data type has been specified, validations and help text from the data type are copied. If element notes were created, they are transferred as field-level help text.

Validation rules in the repository are flagged as either "coming from the Logical Modeler" or being "added afterwards". Validation rules added afterwards (i.e. manually in LANSA) are not lost when the elements are rebuilt in the repository.

Build Database File

The database build creates a database file for the entity. The fields within the file will be based on the elements within the entity. Logical views for the file will be based on relationships and user views.

The database build will also create new elements in the repository based on the physical file layout. For example, any inherited elements must be defined to the repository before the physical file can be defined.

Depending on entity type, a database build may not be allowed – Conceptual entities cannot be database built.

All repository field builds should have been completed before the database builds are submitted.

Build Relationships

The relationships build is responsible for creating the Access Routes and Referential Integrity validations which result from relationships. For example, a join relationship will result in the creation of an Access Route and a file look-up validation rule.

Relationship builds apply to entities which are related to other entities. Relationship builds cannot be performed if the related entities have not been database built.

Depending on entity type, a relationship build may not be allowed – Conceptual entities cannot be relationship built. All database builds should have been completed before the relationship builds are submitted.

10.2 Build Methods

Logical Modeler builds can be performed using one of three methods:

- selecting individual entities,
- building multiple entities by analyzing their build status or
- building all entities, regardless of their build status.

Each method serves a particular need in the modeler.

If you require that only a very small part of the model be built, individual entities can be selected for build. If you wish to build the entire model after a large number of changes, either the entire model or only those entities that are required to be built can be analyzed and the appropriate build acitivities sequenced automatically on the builder queue.

Abstractions can be used to control the entities selected for the build. For more details, refer to 8. Abstractions and 8.3 Work with Abstractions.

10.3 Understanding Builds

Following is a summary of what is built in LANSA for each part of your model:

- Builds and Entities
- Builds and Elements
- Builds and Relationships
- Builds and User Views

Builds and Entities

Data and Variant entities are implemented in the repository as database tables. The table name is taken from the entity physical file name.

Builds and Elements

Each element is implemented as a repository field. It also becomes a field in a file during the database build.

The identifying element will become part of the key to the file. The complete file key may be determined by relationships.

If a data type was specified for the element, the field will reference the corresponding data type field's characteristics. In addition, validation rules and help text will be copied from the reference field.

If a data type was not specified, the field is created from the type and length details in the model.

Element notes become field-level help text.

Element field names in the repository will have the format:

XXXXXXXXX

where:

xxxxxxxx is the element field name.

Builds and Relationships

A relationship will create:

- access routes between the two files which it relates
- validation rules for referential integrity between the two files which it relates
- a logical file for the file which includes the inherited, foreign key element(s).

If a logical view name has been specified on the relationship, it will be used. Note that a logical view name should have been specified if the physical file name of the affected entity is greater than the maximum length allowed (10 if the *RPGIV setting in your IBM i enabled, 8 if it is not). A build error will be generated if the entity physical file name is longer than the maximum allowed and a logical view name has not been specified.

If the physical file name is up to two characters shorter than the maximum number of characters allowed, the modeler will create one automatically, with a name in the format:

ooooooonn

where:

00000000 is the entity physical file name and

nn is a unique, two-digit number.

Builds and User Views

A user view will create a logical file in LANSA.

If a logical view name has been specified, it will be used. Note that a logical view name should have been specified if the physical file name of the affected entity is greater than the maximum number of characters allowed. A build error will be generated if the entity physical file name is more than the maximum number of characters allowed and a logical view name has not been specified.

If the physical file name of the affected entity is up to two characters less than the maximum length allowed and a logical view name has not been specified, the modeler will create one automatically, with a name in the format:

ooooooonn

where:

00000000 is the entity physical file name and

nn is a unique, two digit number. Note that these numbers will follow on sequentially from those in logical files which have been automatically created by the modeler as the result of relationships with other files.

10.4 Rebuilding Your Model

When you rebuild parts of your model, it is important to understand the impact on the LANSA Repository. For example, what happens when you delete an entity? Or what happens if you change an element definition?

When you remove entities from your model, the corresponding file in the LANSA Repository is NOT removed. For example, if you delete a Customer entity, the related fields and file will not be removed from the repository.

When an element is rebuilt, it will replace the existing definition. For example, any manual changes to help text would be lost when an element is rebuilt. It is important that you implement proper procedures during the modeling stage of the project to provide the maximum benefit from the model builds.

Therefore, it is recommended that you do not perform any manual modification of your database until you have completed all possible work with the Logical Modeler.

Once you reach a point where you are completely satisfied with the modeler implemented physical database, you can start to prototype your applications and enhance the database with LANSA features such as virtual fields, predetermined joined fields, etc.

Once you start to develop your database and applications using the LANSA development environment, you should effectively discard your model. Any changes to the model and its subsequent rebuilding will overwrite any changes you have made manually.

The Logical Modeler supports a one-way build process. Model changes can be made and the database can be rebuilt. The modeler is not designed for a twoway build process where changes to the physical model are read back into the logical model. Because the build is a one-way process, you must carefully consider when to end your logical modeling in order to start your physical development.

Remember, minor database changes can be made directly to the physical model. You do not have to make all changes in the logical model and then rebuild. Determining the point at which you should start full development of the database and application should be based on your project plan. Take advantage of the productivity of model process but do not overall analyze or over model your database.

10.5 The Builder Queue

The Builder Queue provides a working list to control the sequencing and execution of the build jobs. It is similar to an iSeries job queue in its functionality: jobs can be deleted, held and released.

To access the Builder Queue, click the ^{see} button on the main toolbar. The Builder Queue window is displayed:

Name Build activity Build status Address Build fields Ready to build Warehouse Build fields Ready to build Product Build fields Ready to build Supplier Build fields Ready to build Customer Build fields Ready to build Order Build fields Ready to build Product Stock Build fields Ready to build Ordered Product Build fields Ready to build Warehouse Build fields Ready to build Varehouse Build fields Ready to build Varehouse Build fields Ready to build Supplier Build file Ready to build Supplier Build file Ready to build Customer Build file Ready to build
Warehouse Build fields Ready to build Product Build fields Ready to build Supplier Build fields Ready to build Dustomer Build fields Ready to build Order Build fields Ready to build Product Stock Build fields Ready to build Drdered Product Build fields Ready to build Manufactured Product Build fields Ready to build Dther Vendor Product Build fields Ready to build Warehouse Build field Ready to build Product Build fields Ready to build Supplier Build field Ready to build
Build fields Ready to build Customer Build fields Ready to build Drder Build fields Ready to build Yroduct Stock Build fields Ready to build Ordered Product Build fields Ready to build Anufactured Product Build fields Ready to build Uther Vendor Product Build fields Ready to build Varehouse Build fields Ready to build Yarehouse Build file Ready to build Yorduct Build file Ready to build Yorduct Build file Ready to build
Build fields Ready to build Inder Build fields Ready to build roduct Stock Build fields Ready to build Indered Product Build fields Ready to build fanufactured Product Build fields Ready to build ther Vendor Product Build fields Ready to build Varehouse Build fields Ready to build varehouse Build file Ready to build upplier Build file Ready to build
Inder Build fields Ready to build roduct Stock Build fields Ready to build roduct Orduct Build fields Ready to build lanufactured Product Build fields Ready to build ther Vendor Product Build fields Ready to build /arehouse Build file Ready to build roduct Build file Ready to build upplier Build file Ready to build
roduct Stock Build fields Ready to build rdered Product Build fields Ready to build lanufactured Product Build fields Ready to build ther Vendor Product Build fields Ready to build /arehouse Build field Ready to build orduct Build field Ready to build upplier Build file Ready to build
rdered Product Build fields Ready to build anufactured Product Build fields Ready to build ther Vendor Product Build fields Ready to build /arehouse Build file Ready to build roduct Build file Ready to build
Ianufactured Product Build fields Ready to build ther Vendor Product Build fields Ready to build /arehouse Build file Ready to build roduct Build file Ready to build policy Build file Ready to build roduct Build file Ready to build upplier Build file Ready to build
ther Vendor Product Build fields Ready to build /arehouse Build file Ready to build roduct Build file Ready to build upplier Build file Ready to build
Varehouse Build file Ready to build roduct Build file Ready to build upplier Build file Ready to build
Varehouse Build file Ready to build roduct Build file Ready to build upplier Build file Ready to build
upplier Build file Ready to build
Error Messages Date Time Message

The top portion of the window shows a sequenced list of jobs that have been queued by either the Build Entity or the Build Model functions. Jobs can have a status of Ready, Ended, Held or Failed. They are initially submitted to the queue in a Ready status.

When an entry is selected in the job list, any error messages that have been generated by that job are shown in the bottom portion of the window.

The Builder Queue window can be resized to increase the number of build jobs and error messages displayed. The split between the top and bottom panes of the builder queue window can also be adjusted.

10.5.1 Delete, Hold and Release Jobs

To delete jobs from the queue, select one or more entries and click the key button. All selected jobs will be deleted from the list.

To hold jobs that are currently in a Ready status, select them in the list and click the set to Held.

To release jobs that are currently in a Held status, select them in the list and click the status. Each job's status will be set to Ready.

10.5.2 Execute Build Jobs

Jobs that are in a Ready status can be executed by clicking the ^{so} button. The following dialog will appear, providing you with options for the build:

Build Options		×
Generate referential integrity val	lidation rules	
	ОК	Cancel

This dialog can be used to control whether the build will generate referential integrity validation rules and whether a database diagram will be generated so that the resultant database can be viewed using the Database Diagram Viewer.

Press Cancel to cancel the build and return to the Builder Queue.

Press OK to continue with the build process.

While build jobs are executing, you are unable to perform other tasks in the modeler.

An executing job has a status of Active. Once it has completed, its status will be set to Ended or Failed, according to the result of the job. The build status of the corresponding entity is also updated. If the job has failed, selecting it in the list will show error messages at the bottom of the window.

10.5.3 Purge the Builder Queue

During the course of a modeling session, the number of entries on the Builder Queue can grow very large. You can purge entries from the queue by clicking the 🗊 button. The Purge Build Queue dialog is displayed:

Purge Build Queue	
🗖 All	
Ready to build	
🔽 Built	
Failed	
Held	
Purge	Cancel

If you do not wish to purge the queue, select *Cancel* otherwise select the statuses you wish to purge and select *Purge*. Queue entries with matching statuses will be deleted from the queue.

10.6 Build Tips and Techniques

Following are some tips and techniques when building your logical model:

Task Tracking and Builds

When you logon to Visual LANSA, you specify a task tracking ID. You must be sure that you are using the same task ID when you use the model build functionality otherwise the builds may fail. For example, if you create objects using one task ID and try to rebuild using a different ID, an error will result as you may not be authorized to replace the existing objects.Site Standards

Before you begin to enter any of the field names, physical file names, logical view names or details for fields, entities, relationships, etc. you should define a set of site or corporate naming standards to be used for your model and the physical database. For example, define the standards to be used for relationship suffixes and prefixes.

Adding Help Text in LANSA

Help text is regenerated whenever the model is rebuilt. If you plan to regenerate your prototype and need to add or change help text, always make changes in Logical Modeler rather than in LANSA, to ensure that the new build includes the changes.

Adding Virtual Fields in LANSA

Virtual fields can be defined in files after the database has been built. It is recommended that you do not add virtual fields until you have completed finished with your logical model. Refer to 10.4 Rebuilding Your Model.

Building with Abstraction

Remember that active abstraction will impact the builder queue. Only objects within the active abstraction are processed.

Deleting Objects

When a built entity is deleted, the related fields, files and so on, are not deleted from the LANSA Repository. You must manually delete these objects.

11. Troubleshooting

This section aims to answer problems that may be experienced with the Logical Modeler.

My model will not load due to an XML error

XML errors should only be encountered if the XML of your model has been manually altered and invalid syntax has been used. You may need to contact your local Product Support team to resolve this issue.

It is strongly recommended that you do not manually alter the XML of a model.

I cannot create a Join relationship to a Variant entity

A Variation relationship must exist for a Variant entity before you can create Join relationships using that Variant entity. Once the Variation relationship is created, the identifying elements are known in the Variant entity and a Join relationship can now be created.

Tutorials

What are the Logical Modeler Tutorials?

The Logical Modeler Tutorials are a set of exercises designed to introduce and reinforce the fundamental logical data modeling skills required to implement a database in LANSA. The tutorials are integrated into the online documentation.

The following tutorials are included:

LGM001 - View a Model

LGM002 - Create an Entity

LGM003 – The Join Relationship

LGM004 – Conceptual Entities and the Includes Relationship

- LGM005 The Parent/Child Relationship
- LGM006 The Variant Entity and the Variation Relationship
- LGM007 User Views

LGM008 – Abstractions

LGM009 – Build the Model

Who Should Use the Tutorials?

Tutorials can be used by novice or experienced LANSA developers who wish to learn how to logically model databases. No LANSA Repository or LANSA RDML skills are required to use the Logical Modeler. You should be familiar with basic modeling concepts such as entities, elements and relationships. Familiarity with the LANSA development environment is an asset.

How Do I Use the Tutorials?

It is recommended that you complete the Tutorials in sequence.

To allow for more than one developer to use the tutorials, all LANSA object names will be prefixed with iii. You may use any three characters, such as the initials of your name, for the iii characters. For example, if you name is John David Smith you can use the characters JDS. Always remember to replace iii with your unique 3 characters.

How Many Developers Can Use the Training?

There is no limit on the number of developers who may use the training at the same time. However, it is important that each developer has a unique identifier for their work.

Your Feedback

Your feedback regarding these tutorials will help us improve the overall quality of the LANSA documentation and training. Please email your comments to lansatraining@LANSA.com.au

LGM001 - View a Model

Objective:

- To execute the Logical Modeler and view an existing model.
- To learn how to use the Logical Modeler interface.
- To highlight the different views of a model.

To achieve the tutorial objectives, you will complete the following steps:

- Step 1. Start the Logical Modeler
- Step 2. Open an Existing Model
- Step 3. Use Pan and Zoom
- Step 4. Change Views
- Step 5. Activate and Deactivate Abstractions
- Step 6. View Entity Elements
- Step 7. Print a Model
- Summary

Before You Begin:

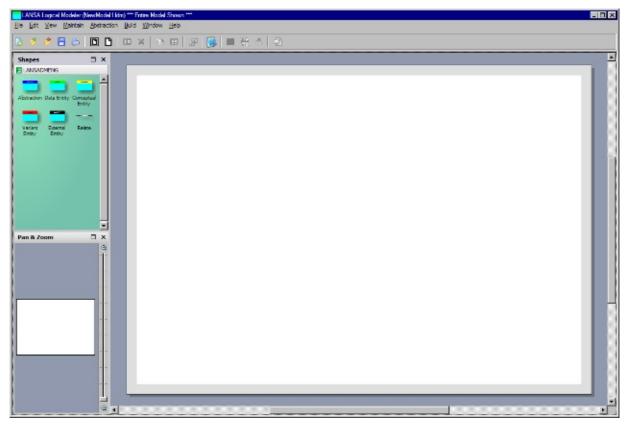
You may wish to review the following topics in the Logical Modeler Guide:

- Getting Started
- Models.

Step 1. Start the Logical Modeler

In this step, you will logon to Visual LANSA and start the Logical Modeler.

- 1. Start Visual LANSA and logon to the DEM partition (recommended).
- 2. Select the *Tools* menu and choose the *Logical Modeler* option.
- 3. The Logical Modeler Main Window should appear, as below.



- 4.If the Document Stencil (Shapes) showing the different entities does not appear, simply use the *View* menu and select the *Show Document Stencil* option.
- 5.If the Pan & Zoom window does not appear, simply use the *View* menu and select the *Pan and Zoom Window* option.

Step 2. Open an Existing Model

In this step, you will open the example model that was shipped with the Logical Modeler.

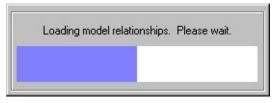
1. Click the Pile Open button on the toolbar or select the *File* menu and choose the *Open*... option.

The Open Model dialog will be displayed:

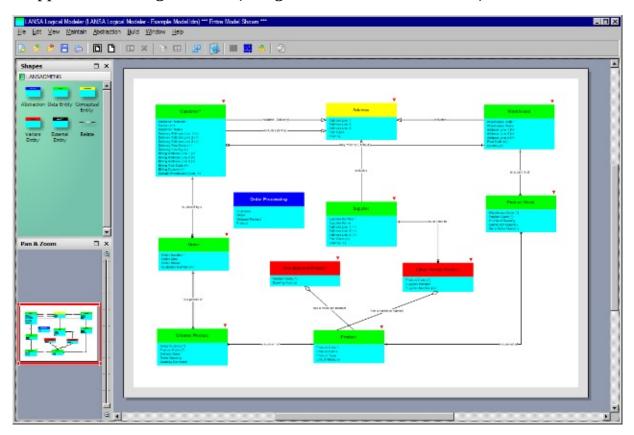
Open Model		? ×
Look in	: 🔁 LANSA 💽 🕝 🤌 📂 🖽 -	
My Recent Documents Desktop My Documents	imports ☐ lansa ☐ k_ide ☐ XSLEditor ■ LANSA Logical Modeler - Example Model.ldm	
here		
there	File name: Ope Files of type: LANSA Logical Data Model (*.ldm) Can	

Note that only files with the '.ldm' suffix are displayed. These are LANSA Data Model files.

- 2. Open the file named 'LANSA Logical Modeler Example Model.ldm' by double-clicking its entry in the list or by selecting it and clicking the Open button.
- 3. The model will begin to load. The following progress bar will be displayed while the model is loading:



The example model is of a simple Sales Order Processing database. It should appear something like this (using the Show Elements view):



Step 3. Use Pan and Zoom

In this step, you will learn how to use the Pan and Zoom facility. The Pan and Zoom facility is used to control what part of the model is displayed.

At this point, a diagram of the entire model should be displayed. Although objects within the model have text associated with them, the text can be hard to read, due to its small size. Using the Pan and Zoom facility can greatly increase the model's readability.

- 1. Click the D Whole Page button on the toolbar or select the *View* menu and choose the *Whole Page* option. This will ensure that the whole model is displayed, enabling you to work easily with the Pan and Zoom facility. Note that the Pan and Zoom window displays a diagram of the entire model at all times.
- 2. Position the cursor slightly above and to the left of the entity in the top-left corner of the Pan and Zoom diagram. Click and hold down the left mouse button.
- 3. While holding down the mouse button, drag the mouse down and to the right. You will see a red border being stretched by the mouse movement. Stop when the red border is approximately half the width of the Pan and Zoom diagram.

Release the mouse button. The diagram will be updated appropriately. The diagram will show the objects surrounded by the red border, similar to the following example:

LANSA Logical Modeler (LANSA Log File Edit Mew Maintain Abstraction	ical Modeler - Example Model Jdm) *** Er	ntire Model Shown ***	
		🎐 🔲 🔛 🌭 🖉 🥹	
Shapes X			· · · ·
	Customer*		Address
Abstraction Data Entity Conceptual Entity	Customer Number * Credit Livit Customer Name Derivery Address Line 1 (>)	-includes (Belivery)-	Address Line 1 Address Line 2 Address Line 2 Address Line 3 Post Code Granty
Variant External Relate	Derivery Address Line 3 (>) Derivery Address Line 3 (>) Derivery Foxt Code (>) Derivery Country (>) Billing Address Line 1 (>)	•	may refer to (Default)
Entity Entity	Billing Address Line 2 (2) Billing Address Line 2 (2) Billing Post Code (2) Billing County (2) DeSuit Warehouse Code (4)		inclutes
Pan & Zoom X		Order Processing	
@	is placed by a	Customer Order	Supplier
	Order	Ordered Product Product	Supplan Namber* Supplan Namber* Address Line 3 (>> Address Line 3 (>> Address Line 3 (>> Post Code (>) County (>)
	Order Number * Order Date Order Status Customer Number (*)	Manufactured	Product
a	۱ ۱	Product Code (*)	Prr 1

- 4. Once a zoom has been activated in this way, the red border can be moved around the diagram in the pan and zoom window, preserving the zoom magnification level.
- 5. To try this, position the cursor in the middle of the red border. A cross-hair cursor is displayed. Hold down the left mouse button and move the mouse to drag the red border to another part of the Pan and Zoom diagram. Release the mouse button to update the model diagram.

Tip:The Pan & Zoom Window can be undocked

Step 4. Change Views

In this step, you will learn how to toggle the view of the model to show Entity Elements or Build Status. These views are important at different times in the modeling process.

- 1. Use the Pan and Zoom function to display as much of the model as possible while maintaining its readability.
- 2. Click on a blank portion of the diagram. This ensures that no diagram objects are selected. Select the *View* menu and choose the *Show Build Status (All Entities)* option. All entities in the diagram will be updated to show their Build Status.

Note that for Data and Variant entities, a status is shown for each of the build phases: Fields, File and Relationships. For Conceptual entities, which are not built into database tables, only the status of the Fields build phase is shown.

- 3. To switch back to the Elements view for all entities, select the *View* menu and choose the *Show Elements (All Entities)* option. As well as being able to change the view for the entire model, individual entities can have their view changed. Ensure the Elements view for all entities is active.
- 4. Select the Customer and Order entities by holding down the shift key and clicking on them. Note that when a shape is selected, it is bordered by gray padlock images. Select the *View* menu and choose the *Show Build Status (Selected Entities)* option. The Customer and Order entities will have their Build Status view displayed, while all other entities retain their Elements view.
- 5. Switch back to the Elements view for the Customer and Order entities by selecting the *View* menu and choosing the *Show Elements (Selected Entities)* option. In some instances, you may wish to show neither the Elements or Build Status views. The modeler enables you to collapse entities, so that only their names are shown. Ensure the Elements view for all entities is active.
- 6. Select the *View* menu and choose the *Show Collapsed (All Entities)* option. All entities in the diagram should have their shapes collapsed, such that only their names are shown. It is also possible to collapse an individual entity. Ensure the Elements view for all entities is active.
- 7. Select the Customer entity by clicking on it.
- 8. Select the View menu and choose the Show Collapsed (Selected Entities)

option. The Customer entity should be collapsed, showing only its name.

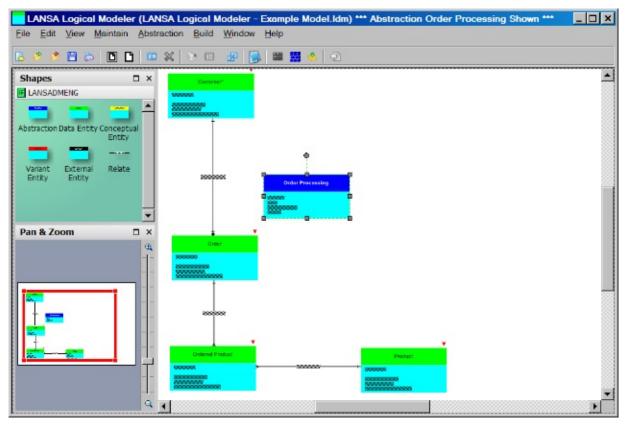
By using combinations of the View options described above, the view of the diagram can be tailored to your specific needs, depending on the phase of your modeling.

Step 5. Activate and Deactivate Abstractions

In this step, you will learn how activating and deactivating abstractions can be used to change the view of the model.

- 1. Activate the Order Processing abstraction. This can be done by doubleclicking it or by selecting it and taking the Activate option from the rightclick or *Abstraction* menus.
- 2. The diagram will be updated to show only the Customer, Order, Ordered Product and Product entities. Note that the Order Processing abstraction is still shown on the diagram.

Note that the navigation facilities you have learned to use in the previous steps of this exercise can be used when an abstraction is active.



- 3. Move the entities around the diagram.
- 4. Deactivate the abstraction. This can be done by double-clicking it or by selecting it and taking the Deactivate option from the right-click or *Abstraction* menus. Note that the entities return to their original positions.
- 5. Reactivate the abstraction. Note that the abstracted entities return to the

positions you moved them to when the abstraction was last active.

- 6. Deactivate the abstraction.
- 7. The diagram will be updated to show the entire model.

Step 6. View Entity Elements

In this step, you will learn how to view an entity's elements in order to work with them.

- 1. Select the Customer entity by clicking on it.
- 2. To view its elements, select the *Elements*... option from its right-click pop-up menu or select the *View* menu and choose *Elements*... option.

The Customer entity elements will be displayed:

Maintain Entity Bements (Customer)				
🤌 🔁 🗶 🗇 🛷 🖉 🗒 🕤				
Name	Data type	Field type/length	Field name	From entity
x Customer Number	🚺 ID Number	Integer (4)		
x Credit Limit	Amount	Packed (9,2)		
× Customer Name		Alphanumeric (50)		
(>) Delivery Address Line 1	Name	Alphanumeric (25)		Address
(>) Delivery Address Line 2	Name	Alphanumeric (25)		Address
(>) Delivery Address Line 3	Name	Alphanumeric (25)		Address
x(>) Delivery Post Code	Zip/Post Code	Alphanumeric (5)		Address
×(>) Delivery Country	Name	Alphanumeric (25)		Address
x(>) Billing Address Line 1	Name	Alphanumeric (25)		Address
x(>) Billing Address Line 2	Name	Alphanumeric (25)		Address
(>) Billing Address Line 3	Name	Alphanumeric (25)		Address
x(>) Billing Post Code	Zip/Post Code	Alphanumeric (5)		Address
x(>) Billing Country	Name	Alphanumeric (25)		Address
x(<) Default Warehouse Code		Alphanumeric (10)		Warehouse

You will learn how to maintain an entity's elements in later exercises.

3. Click the Close button to close the elements window.

Step 7. Print a Model

In this step, you will learn how to print a model diagram.

1. Click the Series Print button on the toolbar, or select the *File* menu and choose the *Print*... option.

The Visio Print dialog will be displayed:

Print			×
Printer			
<u>N</u> ame:	\\NT1\XEROX	▼ Prop	erties
Status:	Ready		
Type:	Xerox DocuPrint 4517		
Where:	10.2.0.200:PASSTHRU	🗖 Cole	or as <u>b</u> lack
Comment		Prin	it to fi <u>l</u> e
Page range		Copies	
⊙ <u>A</u> ll		Number of copies:	1 -
C Curr <u>e</u>	ent page		
C Page	s: from 1 to 1	11 22 33	
C <u>S</u> elec	tion		
2		ОК	Cancel

This dialog behaves like a standard Windows Print dialog.

- 2. Select the printer to which you wish to print by using the drop-down box at the top of the form.
- 3. Press *OK* to print the model.
- 4. Close the model. Do not save your changes.

Summary

Important Observations

- The Logical Modeler is a very mouse-driven application, due to its graphical nature. Right mouse button clicks and drag-and-drop facilities are used to a great extent.
- The Logical Modeler provides many different ways of accomplishing a given task. Actions can be performed by right-clicking on a diagram shape, by selecting an option from the menus, or by clicking a button on the toolbar.
- Note that different options are enabled in the menus and the toolbar depending on the type of object selected in the diagram.

Tips & Techniques

- Remember that you can show a particular view for all entities or selected entities. In addition, you can choose to not display a view at all, and show entities in their collapsed state.
- Abstractions are a good way of grouping entities in a large model. The model diagram can be simplified by activating an abstraction and moving the abstracted entities. Remember that build activities will be restricted to the objects within the active abstraction.
- A new page is created within the diagram every time a new model is opened. When printing a model, ensure the Current Page option is selected to avoid printing all open models.

What I Should Know

- How to start the Logical Modeler.
- How to open an existing model.
- How to zoom in to view different parts of a model diagram.
- How to use the different views to change how the model is displayed.
- How to work with an entity's elements.
- How to Activate and Deactivate Abstractions.
- How to print a model diagram.

LGM002 - Create an Entity

Objective:

- To create a new model which will match part of the sample model viewed in the first tutorial. You will begin by creating the Customer entity.
- To learn how to create an entity.
- To learn how to define and manipulate elements within an entity.
- To learn how to create user notes.
- To learn how to save a model.

To achieve the tutorial objectives, you will complete the following steps:

- Step 0. Start the Logical Modeler
- Step 1. Create a Data Entity
- Step 2. Define Elements for the Entity
- Step 3. Work with Elements in Entities
- Step 4. Assign an Identifying Element
- Step 5. Save the Model
- Summary

Before You Begin:

You may wish to review:

- Introduction to Data Modeling
- Entities
- Elements
- Data Types
- Saving a Model.

In order to complete this tutorial, you must have completed the following:

• LGM001 - View a Model

Step 0. Start the Logical Modeler

In this step, you will start the logical modeler (if has not already been started) and create a new model.

1. Start the Logical Modeler.

If the modeler is already started, click the **S** New button on the main toolbar, or select the *File* menu and choose the *New*... option.

2. A blank model diagram will be displayed. You are now ready to begin creating your new model.

Step 1. Create a Data Entity

1. Drag the Data Entity shape from the Document Stencil onto the model diagram.

The Create Data Entity dialog will be displayed:

Create Data Entity			
Entity name			
Physical file name			
		OK	Cancel

2. Enter an Entity name of Customer.

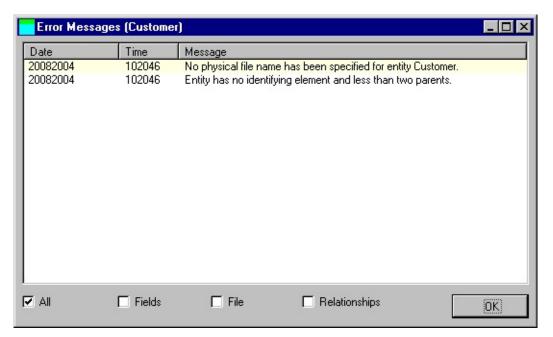
Do not specify a physical file name for the entity at this stage.

3. Click the *OK* button.

The shape you dropped onto the diagram will be updated and will appear something like this:



- 4. Note the inverted red triangle to the top-right of the shape. Whenever you see this triangle, it means that the entity has error messages.
- 5. Right-click the Customer entity and choose the Error messages... option from the pop-up menu. The following dialog will be displayed:



Note that dynamic validation of the entity has taken place; you do not have to wait until you build your model in order to see errors that will prevent it from being built.

From time to time in the remaining tutorials, when you see the red triangle, use the Error messages... option to see some of the different errors that are generated for an entity.

Step 2. Define Elements for the Entity

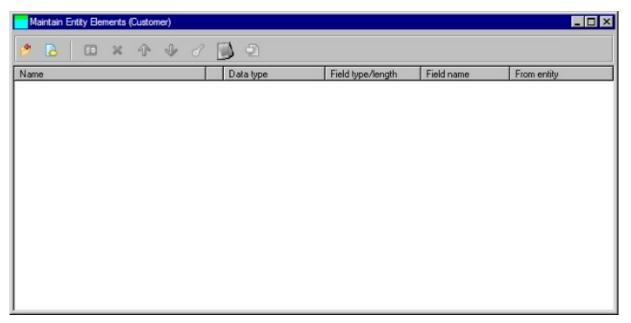
In this step, you will add the elements for the Customer entity. Some elements will be defined using Data Types and some elements will be defined manually.

The Customer entity will have the following elements: Customer Number, Customer Name, Address Line 1, Address Line 2, Address Line 3, Address Line 4, Postal Code and Country.

Note: If you compare the Customer Entity to the one in the sample, you will see that it has a different structure. In tutorial LMG004, you will transfer some of these elements to a Conceptual Entity and use relationships to include them into the Customer Entity so that it matches the sample model.

1. Right-click the Customer entity and select the *Elements*... option.

The Maintain Entity Elements window will be displayed:



Note that this window can also be displayed by selecting the entity and then selecting the *Edit* menu and choosing the Entity Elements... option.

2. Click the 🔼 New button to create a new element.

The Create Element dialog will be displayed:

Create Element		
Element name		
Definition		
O Data type	Amount	•
◯ Type/length	Packed	Length Dec
Field name		
		OK Cancel

- 3. Enter a Name of Customer number.
- 4. Using the Data type drop-down list, select the *ID Number (Long)* data type. Note that the type and length of the data type is displayed to the right of its description. This can aid you in selecting an appropriate data type.
- 5. Give the element a field name of iiiCUSNUM, where iii are your initials.
- 6. Click the *OK* button.
- 7. The Customer Number element will be added to the list of elements and the Create Element window will be redisplayed.
- 8. Create an element called Customer name. This field will be manually defined as follows:
 - a. Select the *Type/length* option.
 - b. Using the drop-down list, select *Alphanumeric* and specify a length of 50.
 - c. Enter a field name should be iiiCUSNAM.
 - d. Click the *OK* button.
- 9. Create the remaining element for the Customer entity as follows:

Element Name	Data Type	Field Name
Address line 1	Text	iiiADDL1
Address line 2	Text	iiiADDL2
Address line 3	Text	iiiADDL3
Address line 4	Text	iiiADDL4
Post code	Zip/Post Code	iiiPSTCD

Country Description (Short) iiiCNTRY

Note that, although you have specified field names for all the elements you have created, it is not necessary. The field name is only required when the entity is built.

10. Click the *Cancel* button to return to the entity elements window.

Step 3. Work with Elements in Entities

In this step, you will change the Customer Name element to user a data type. You will also learn how to change the sequence of the elements in an entity. The sequence of elements is important as it determines the sequence of the fields in the file when an entity is built. Finally, you will remove an element from the entity.

1. Double-click the Customer Name element and change it to use the *Name* data type.

Click the *OK* button.

Its entry will be updated in the elements list to show its new definition.

2. Select the Country element.

Click the ¹ Move Up button on the toolbar. The element moves above Post Code in the list.

Click the 🎐 Move Down button to move it below Post Code again.

Note that the sequence in which the elements appear here will be the sequence of the fields in the file when the entity is built. Try re-sequencing other elements.

3. In your model, you may decide that Address Line 4 is no longer needed. To remove it, select Address Line 4 and click the 🔀 Delete toolbar button.

You will be asked whether you wish to perform the deletion.

Click the *Yes* button. The element will be removed from the list.

Note that the element is not actually deleted at this stage. It is merely detached from the element. Should it be required at a later stage, it can be accessed from the Unattached Elements window.

Step 4. Assign an Identifying Element

In this step, you will create the Identifying Element for the entity. Only one element may be selected as the identifying element. For the Customer entity, the Customer Number is the identifying element.

1 Select the Customer Number element and click the **Set** Identifier button on the toolbar.

Note that a key image appears next to the element. This denotes that Customer Number is the identifying element of Customer.

2. Select the Customer Name element and click the *set* Identifier button.

Note that the key image is removed from Customer Number and assigned to Customer Name.

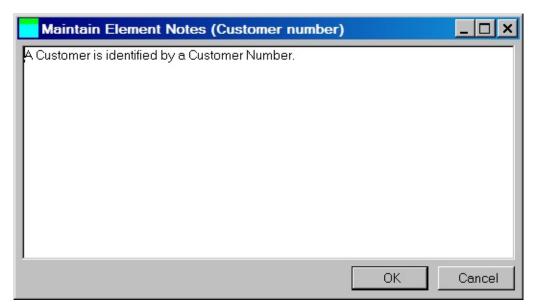
- 3. Click the definition Set Identifier button again the key image is removed from Customer Name.
- 4. Select Customer Number again and click the *Set* Identifier button to make it the identifying element of Customer.

At this stage, the Maintain Entity Elements window should appear something like this:

Number (Long) kt kt	Packed (15,0) Alphanumeric (50) Alphanumeric (50) Alphanumeric (50)	LPCCUSNAM LPCADDL1	
ĸt	Alphanumeric (50)	LPCADDL1	
ĸt			
	Alphanumeric (50)	I DOLD DI D	
		LPCADDL2	
ĸt	Alphanumeric (50)	LPCADDL3	
et .	Alphanumeric (50)	LPCPSTCD	
ĸt	Alphanumeric (50)	LPCCNTRY	
	×t		

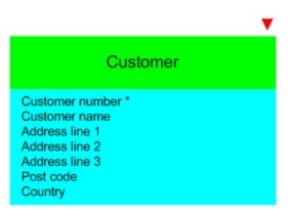
5. Select the Customer Number element and click on the Source Notes toolbar icon. The Maintain Elements Notes dialog will be displayed.

a. Add some help text to the Customer Number.



- b. Press *OK* to close the dialog.
- 6. Click the 🖻 Close toolbar button to close the Customer Entity Elements window.

The Customer diagram shape will be updated and should look something like this:



Note that Customer Number is suffixed with an asterisk. The * denotes that Customer Number is Customer's identifying element.

- 7. Try moving the Customer shape around the diagram.
- 8. Use the Pan and Zoom facility to enlarge the view of the diagram.
- 9. Switch between the Elements, Build Status and Collapsed views for Customer.

Note that there is an inverted red triangle above the top-right of the entity. This

indicates that the entity has validation errors.

Step 5. Save the Model

In this step, you will save the model.

1. To save the model, click the 🔲 Save button on the main toolbar, or select the File menu and choose the *Save* option.

The Save Model As dialog will be displayed:

Save Model As		? ×
Save in	: 🗀 LANSA 💽 🕝 🤌 📂 🖽 -	
My Recent Documents Desktop My Documents	 Imports Iansa Ik_jde XSLEditor ▲ LANSA Logical Modeler - Example Model.ldm 	
there	File name: NewModel1.ldm Save as type: LANSA Logical Data Model (*.ldm) Cancel	

2. Enter a name that is meaningful to you and click the *Save* button.

The model diagram will be redisplayed and a Model Saved message will be displayed at the bottom of the main window.

Summary

Important Observations

- Creating an entity is as simple as dragging and dropping the appropriate entity shape from the Document Stencil onto the model diagram. You will learn more about the different entity shapes in a later tutorial.
- Although field and file names are required when elements and entities are built, they do not need to be specified when the element or entity is first created.
- •
- Note that a *Save As...* option is available on the *File* menu for saving a copy of your model under a different name.

Tips and Techniques

- Create elements within an entity as you think of them. You can re-sequence them at a later stage.
- It is recommended that you use data types when defining a logical model. Data types provide a level of abstraction to your design. They ensure that standards are enforced. For example, all Date fields will have the same formatting.
- The modeler will warn you if you have made changes to a model and are exiting without having saved your work.

What I Should Know

- How to create an entity.
- How to create, change, delete and re-sequence elements for an entity.
- How to set an identifying element for an entity.
- How to create user notes.
- How to save a model.

LGM003 – The Join Relationship

Objective:

- To learn how to create a Join relationship between two entities.
- To understand how Join relationships impact entities.

To achieve the tutorial objectives, you will complete the following steps:

- Step 1. Create an Order Entity.
- Step 2. Relate the Order and Customer Entities.
- Step 3. Sequence Elements in Order Entity
- Summary

Before You Begin:

You may wish to review the following topics in the Logical Modeler Guide:

- Relationships
- Work with Relationships.

In order to complete this tutorial, you must have completed the following:

- LGM001 View a Model
- LGM002 Create an Entity

Step 1. Create an Order Entity

In this step, you will create a data entity called Order. The Order entity will be related to the Customer entity in Step 2 of the tutorial.

- 1. If the model you saved in tutorial LGM002 Create an Entity is not currently displayed, open it by selecting the *File* menu and choosing the *Open*... option.
- Create a data entity named Order.
 Set its physical file name to iiiORDER, where iii are your initials.
- 3. Create the following elements for the Order entity:

Element Name	Data Type	Field Name
Order number	ID Number (Long)	iiiORDNUM
Order date	Date	iiiORDDAT
Order status	Code	iiiORDSTS

- 4. Make Order Number the identifying element of Order.
- 5. Close the elements window.

Step 2. Relate the Order and Customer Entities

In this step, you will create a relationship between the Order and Customer entities and see the impact it has on the Order entity's elements. Our model is based on a classical order processing system where customers place orders for products.

1. Drag the Relate connector from the Document Stencil onto a blank portion of the model diagram.

Note that once dropped, the connector remains as the selected shape, with green handles at each end.

2. Drag-and-drop the Source end handle of the Relate shape (denoted by <- Source) to the centre of the Order entity.

Note that the shape will only be properly connected to the entity when the entity's border turns red.

3. Drag-and-drop the Target end handle of the Relate shape to the centre of the Customer entity. Ensure that the border of Customer turns red before dropping the end of the connector.

If both ends of the connector have been correctly attached, the Create Relationship dialog will be displayed:

Create Relationship
Relationship
Order
is a parent of
Customer
Share inherited identifiers
Descriptors
Order
<u></u>
Customer
Customer
Order
Descriptive prefix
Customer Physical Object
Logical view name
OK Cancel

The relationship you are going to create states that an Order must refer to Customer, i.e. an Order cannot exist without referring to an existing Customer.

4. Select 'must refer to' from the Relationship Type drop-down box at the top of the window.

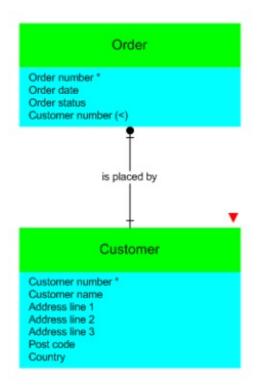
Depending on the types of entity being related, the Relationship Type dropdown list at the top of the window will contain different types of allowable relationships. In this instance, because we are relating two data entities, the allowable relationships are Parent/Child (is a parent of), Mandatory Join (must refer to) and Optional Join (may refer to).

- 5. The descriptor of a relationship allow the user to specify, in easily understandable terms, the nature of the relationship between the two entities. In this instance, they will be defaulted to 'must refer to' and 'must be referred to by'. Change these to 'is placed by' for the Order/Customer descriptor and 'places an' for the Customer/Order descriptor.
- 6. Note that the Logical View Name entry field, toward the bottom of the dialog, has been enabled. This is because that, as a result of this join

relationship, a logical view for the Order file will be created by the modeler when the model is built. Note that the caption of the box surrounding the entry field has also changed to show to which entity (Order) the logical view name will belong when created. Also shown is the entity's physical file name, to aid you in determining what the logical view name should be called.

Leave the logical view name blank. The impact of this will be shown later.

- 7. The field names of the inherited elements are important when the entities are built –they must be unique. This will be explained more fully in the following tutorial: LGM004 Conceptual Entities and the Includes Relationship.
- 8. Click the *OK* button. The model diagram will be updated and the Customer and Order entities will appear something like this:



Note the following about the updated diagram:

- The Order entity has inherited the identifying element of Customer Number.
- Order's Customer Number is denoted as being inherited via a join relationship with a lesser than sign in brackets (<).
- The connector between the two entities has had its text set to the Source to Target description that you entered.

• The full impact of the relationship on the two entities will be shown and discussed further in a later tutorial, where these entities will be built.

Step 3. Sequence Elements in Order Entity

In this step, you will see how the relationship has impacted the Order Entity. You will sequence the elements so that the Customer Number is the second element listed in the entity.

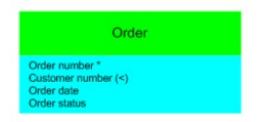
- 1. Display the Maintain Entity Elements window for the Order entity.
 - a. Select the Customer Number element.
 - b. The Maintain Entity Elements window should appear something like this:

	7 💽 🥥 Data type	Field type/length	Field name	From entity
Order number	ID Number (Long)	Packed (15,0)	LPCORDNUM	
Order date	Date	Signed (6,0)	LPCORDDAT	
Order status	Code	Alphanumeric (3)	LPCORDSTS	
Customer number	ID Number (Long)	Packed (15,0)	LPCCUSNUM	Customer

Note the following about the inherited Customer Number element:

- It cannot be specified as the identifying element of the entity.
- Its notes can be maintained.
- It can be re-sequenced in the list.
- It can be deleted (note that deleting the element will delete the relationship with the Customer entity).
- 2. Move the Customer Number element to immediately below the Order Number.

Note that elements can be positioned anywhere within the entity, depending on your preference of what their sequence should be. The sequence of identifying elements will determine the primary key of the physical file once the entity is built. 3. Click the Close button to close the Entity Elements list. The Order entity should now appear something like this:



4. Save your model.

Summary

Important Observations

- When creating a relationship, it will only be recognized if the borders turn red when connecting the Relate connector to the two entities.
- The allowable relationships between two entities depends on the types of entity being related.
- Deleting an inherited element from an entity will delete the entire relationship via which that element has been inherited.
- The model diagram shows inherited elements with a symbol to their right to denote the type of relationship by which they were inherited.

Tips and Techniques

• Change the default descriptors for a relationship to something more meaningful to the nature of the relationship between the two entities.

What I Should Know

- How to create a relationship between two entities.
- How inherited elements are shown on the model diagram and in the Entity Elements window.

LGM004 – Conceptual Entities and the Includes Relationship Objective:

• To learn how to create a conceptual entity.

- To learn how to transfer elements between two entities.
- To learn how to create multiple relationships between a data entity and a conceptual entity.

To achieve the tutorial objectives, you will complete the following steps:

- Step 1. Create a Conceptual Entity
- Step 2. Transfer Elements from Customer to Address
- Step 3. Relate the Customer and Address Entities
- Summary
- Before You Begin:

You may wish to review the following topics in the Logical Modeler Guide:

- Entities
- Transferring Elements
- Relationships.

In order to complete this tutorial, you must have completed the following:

- LGM001 View a Model
- LGM002 Create an Entity
- LGM003 The Join Relationship.

Step 1. Create a Conceptual Entity

Although the Customer entity contains address information, it is now required to hold two addresses: Delivery Address and Billing Address. Instead of specifying more address elements in the Customer entity, you will create an Address conceptual entity which will be related to the Customer entity multiple times.

- 1. Ensure the model you created in the previous tutorial is open.
- 2. Drag the Conceptual Entity shape from the Document Stencil onto the model diagram.

The Create Conceptual Entity dialog will be displayed:

Create Concep	tual Entity		
Entity name	[
		OK	Cancel

Note that a physical file name is not specified for a Conceptual entity – when it is built, a database file is not generated.

3. Name the entity Address and click the *OK* button.

The entity's shape will be updated with its name on the diagram.

Step 2. Transfer Elements from Customer to Address

Because you have already defined some address elements in the Customer entity, you do not need to re-define them. In this step, you will transfer them from the Customer entity to the Address entity.

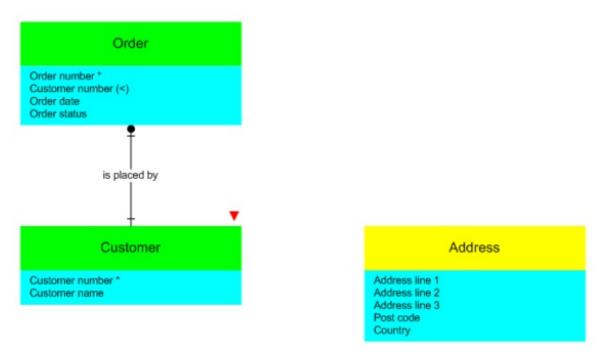
1. Open the Entity Elements window for both the Customer and Address entities.

Position the Address elements window beneath the Customer elements window.

- 2. Using the Shift or Control key, select all of the address elements in the Customer entity.
- 3. Drag the selected elements from the Customer elements list to the Address element list and drop them.

The elements will be removed from the Customer elements list and will be added to the Address elements list.

4. Close both Entity Elements windows. Your model diagram should now appear something like this:



Step 3. Relate the Customer and Address Entities

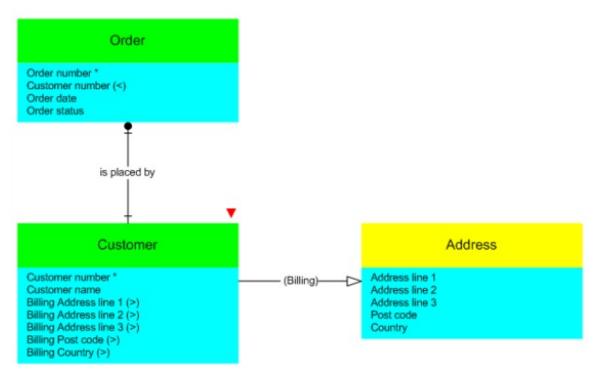
In this step, you will create multiple relationships between the Customer and Address entities in order to support Customer Delivery and Billing Addresses.

1. Define an Includes relationship between Customer and Address.

Customer will be the source entity and Address will be the target entity.

Specify Billing as the Descriptive Prefix. Note that the only allowable relationship between a Data entity and a Conceptual entity is the Includes relationship.

Once the relationship has been defined, the model diagram will appear something like this:

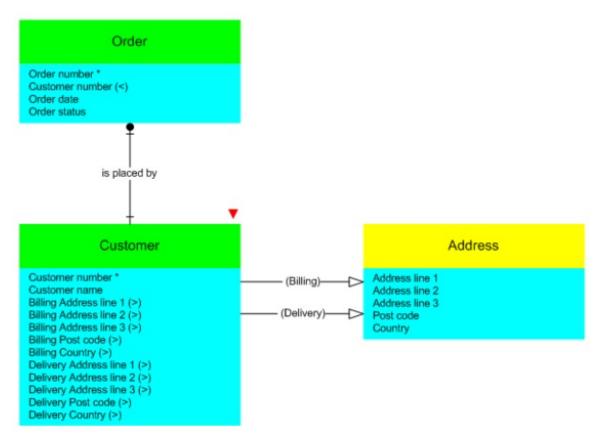


Note the following about the diagram:

- The elements from the Address entity have been inherited by the Customer entity.
- The inherited elements are denoted by a greater than sign in brackets (>) to the right of their names.
- The inherited elements have been prefixed with the Descriptive Prefix of the relationship.

- The Descriptive Prefix has been added to the text of the connector.
- 2. Define another Includes relationship between Customer and Address. Specify Delivery as the Descriptive Prefix.

The model diagram should now appear something like this:



Note that the Descriptive Prefix of a relationship is not mandatory – the relationship between the Order and Customer entities does not have one specified. In the case of the relationships between Customer and Address, however, it is advantageous to specify Descriptive Prefixes, so that the inherited elements can be differentiated.

3. Display the Customer Entity Elements window.

Note that the two sets of elements that have been inherited from the Address entity have the same field names. One of the rules for an entity build is that all field names must be unique within the entity.

4. For each of the 'Delivery' address elements, append a 'D' to the end of their field names.

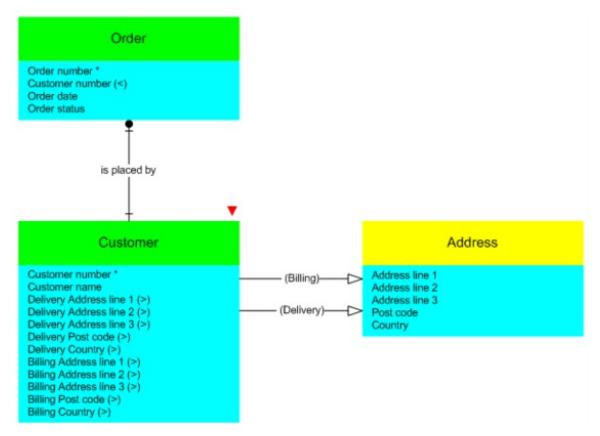
For each of the 'Billing' address elements, append a 'B' to the end of their

field names.

Note that all field names are now unique within the Customer entity.

- 5. Move the Delivery Address above the Billing Address. Notice that all of the included elements move together.
- 6. Close the Customer Entity Elements window.

The model diagram should now appear something like this:



7. Save the model.

Summary

Important Observations

- The Descriptive Prefix can be important in differentiating between inherited elements if there is more than one relationship between the two entities.
- If an element is inherited more than once by an entity, its field name must be unique within the entity.
- When you move the included attributes in the Customer entity, they are moved as a group of attributes.

Tips and Techniques

- If you think you may have created an element in the wrong entity, do not worry as you can transfer it to another entity at a later stage.
- Change the default descriptors for a relationship to something more meaningful to the nature of the relationship between the two entities.
- To resequence the order of the address elements, you must work with the Address entity. If the element order is changed in the Address entity, the order will also be updated in the Customer entity.
- The Address conceptual entity will ensure that all addresses have the same format. If you change the format of the postal code, it needs to be done in only one place in the complete model and all entities will be updated.

What I Should Know

- How to create a conceptual entity.
- How to transfer elements between entities.
- How to create an Includes relationship between two entities.
- The importance of the Descriptive Prefix when creating multiple relationships between the same entities.
- When to change the field name of inherited elements.

LGM005 – The Parent/Child Relationship

Objective:

- To learn how to create a Parent/Child relationship between two entities.
- To understand how Parent/Child relationships impact entities.
- To create relationships between Product and Ordered Product, and relationships between Order and Ordered Product.

To achieve the tutorial objectives, you will complete the following steps:

- Step 1. Create an Ordered Product Entity
- Step 2. Relate the Order and Ordered Product Entities
- Step 3. Create a Product Entity
- Step 4. Relate Product and Ordered Product
- Summary

Before You Begin:

You may wish to review the following topics in the Logical Modeler Guide:

• Relationships.

In order to complete this tutorial, you must have completed the following:

- LGM001 View a Model
- LGM002 Create an Entity
- LGM003 The Join Relationship
- LGM004 Conceptual Entities and the Includes Relationship

Step 1. Create an Ordered Product Entity

In this step, you will create a data entity called Ordered Product. The Ordered Product entity will be related to the Order entity in Step 2 of this tutorial.

- 1. Ensure the model you created in the previous tutorial is open.
- 2. Create a data entity called Ordered Product.

Set its physical file name to iiiOPR, where iii are your initials.

3. Create the following elements for the Ordered Product entity:

Element Name	Data Type	Field Name
Delivery date	Date	iiiDLVDAT
Order quantity	Quantity	iiiORDQTY
Delivered quantity	Quantity	iiiDLVQTY
Backordered quantity	Quantity	iiiBOQTY

- 4. Make Delivery Date the identifying element of Ordered Product.
- 5. Close the elements window.

Step 2. Relate the Order and Ordered Product Entities

In this step, you will create a relationship between the Order and Ordered Product entities and see the impact it has on the Ordered Product entity's elements. Remember, the model is based on a classical order processing system, where customers place orders for products. The relationship you are going to create states that an Order is the parent of Ordered Product, i.e. an Ordered Product cannot exist without its parent entity, Order.

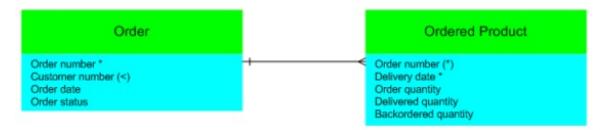
- 1. Drag the Relate connector from the Document Stencil onto a blank portion of the model diagram.
- 2. Connect the relate shape source to the Order entity and connect the target to the Ordered Product entity.
- 3. In the Create Relationship dialog, select 'is a parent of' from the Relationship Type drop-down box.

Leave Descriptive Prefix blank.

Note that Logical View Name is disabled. The impact of this will be shown later.

Click the *OK* button.

4. The model diagram will be updated and the Order and Ordered Product entities will appear something like this:



Note the following about the updated diagram:

- The Ordered Product entity has inherited the identifying element of Order Number, which is now an identifier of Ordered Product.
- The Ordered Product's Order Number is denoted as being an inherited identifier by an asterisk in brackets to its right.
- The full impact of the relationship on the two entities will be shown and discussed further in a later tutorial, where these entities will be built.

Step 3. Create a Product Entity

In this step, you will create another data entity, Product. Some notes will be assigned to the Product Number element and the Product entity will be related to the Order Product entity.

1. Create a Data Entity called Product.

Set its physical file name to iiiPRD, where iii are your initials.

2. Create the following elements for the Product entity:

Element Name	Data Type	Field Name
Product number	ID Number (long)	iiiPRDNUM
Product name	Name	iiiPRDNAM
Quantity on hand	Quantity	iiiQTYOH
Quantity committed	Quantity	iiiQTYCOM

- 3. Make Product Number the Identifying Element of Product.
- 4. Select the Product Number element and click the 🚺 Notes toolbar button. The Maintain Element Notes dialog will be displayed:

Maintain Element Notes (Product number)		_ 🗆 ×
	OK	Cancel

5. Enter some notes for the element: The Product Number uniquely identifies a product within the system. Note that the text entered here is completely free-format.

Click the *OK* button.

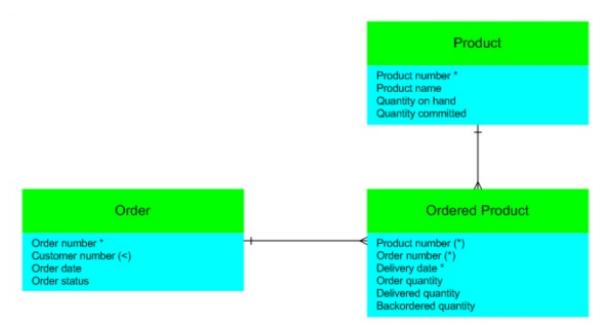
- 5. Note that a notepad image is displayed to the right of the Product Number. This denotes that the element has notes associated with it.
- 6. Close the elements window.

Step 4. Relate Product and Ordered Product

In this step, you will make Product a parent of Ordered Product and manipulate the sequence of Ordered Product's identifying elements.

1. Make Product a parent of Ordered Product. (Be sure to connect the source to the Product entity.)

The Order, Ordered Product and Product entities should appear something like this:



Note that Ordered Product has inherited Product's identifying element, Product Number, as one of its identifiers (denoted by an asterisk in brackets).

Note also that Product Number has become the first identifying element of Ordered Product. However, it is usual in an order processing system for an entity like Ordered Product to have Order Number as its first identifying element, such that data can be easily read from the file in Order Number sequence.

- 2. Open the Maintain Entity Elements window for Ordered Product.
- 3. Move Order Number above Product Number.

Note that, because Order Number and Product Number originate from two different entities, the sequence of the inherited identifying elements can be changed.

The Ordered Product entity should now appear something like this:



4. Display the Change Relationship dialog for the relationship between Order and Ordered Product. Note that the Logical View Name for the relationship is disabled. No logical view will be created to support this relationship; it will be supported by the natural order of the entity's identifying elements.

Display the Change Relationship dialog for the relationship between Product and Ordered Product. Note that the Logical View Name is enabled. This is because the relationship is the second parent/child relationship to affect Ordered Product's identifying elements, so a logical view will be created to support the relationship.

5. Save your model.

Summary

Important Observations

- The identifying elements of the source entity in a parent/child are inherited and become identifying elements of the target entity.
- The inherited identifying elements are denoted by the symbol (*) on the diagram.
- The source and target end of the connector is important in determining which entity will be the parent (source) and which object will be the child (target).
- Subsequent parent/child relationships to the first parent/child relationship in a target entity will generate logical views in the target entity.

Tips and Techniques

- If an entity has more than one parent, define them in any order you wish. The inherited identifying elements can be re-sequenced at a later stage.
- Enter notes against elements. Notes will be transformed into field-level help text when the entity is built.

What I Should Know

- How to create a parent/child relationship between two entities.
- How to re-sequence identifying elements.
- How inherited identifying elements are shown on the model diagram and in the Entity Elements window.

LGM006 – The Variant Entity and the Variation Relationship Objective:

- To learn how to create a Variant entity.
- To learn how to create a Variation relationship between two entities.
- To learn how to differentiate between variants in the source entity of the Variation relationship.
- To understand how Variation relationships impact entities.
- To create the Supplied Product and Manufactured Product variant entities in the model and to create variation relationships to the Product entity.

To achieve the tutorial objectives, you will complete the following steps:

- Step 1. Create a Manufactured Product Entity
- Step 2. Relate the Product and Manufactured Product Entities
- Step 3. Create a Supplied Product Entity and Relate it to the Product Entity
- Step 4. Create a Product Type Element for the Product Entity
- Summary

Before You Begin:

You may wish to review the following topics in the Logical Modeler Guide:

- Entities
- Relationships.

In order to complete this tutorial, you must have completed the following:

- LGM001 View a Model
- LGM002 Create an Entity
- LGM003 The Join Relationship
- LGM004 Conceptual Entities and the Includes Relationship
- LGM005 The Parent/Child Relationship

Step 1. Create a Manufactured Product Entity

In this step, you will create a Variant entity called Manufactured Product. A Variant entity is used to group elements which are part of the Product entity but specific to products which are manufactured. The Manufactured Product entity will be related to the Product entity in Step 2 of the tutorial.

- 1. Ensure the model you created in the previous tutorial open.
- 2. Drag the Variant Entity shape from the Document Stencil onto the model diagram.

The Create Variant Entity dialog will be displayed.

3. Give the entity a name of Manufactured Product.

Set its physical file name to iiiPRM, where iii are your initials.

4. Create an element for the entity called Drawing number.

Select the Number data type.

Set its field name to iiiDRWNUM, where iii are your initials.

Note that an identifying element for the entity cannot be set – Variant entities can only inherit their identifying elements from another entity via a Variation relationship.

5. Close the elements window.

Step 2. Relate the Product and Manufactured Product Entities

In this step, you will create a relationship between the Product and Manufactured Product entities and see the impact it has on the Manufactured Product entity's elements.

1. Drag the Relate connector from the Document Stencil onto a blank portion of the model diagram.

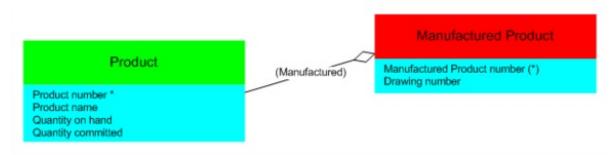
Connect the source end of the relate shape to the Product entity and connect the target end to the Manufactured Product entity. The relationship you are going to create states that Product has a variant called Manufactured Product, i.e. a Manufactured Product is a type of Product.

2. In the Create Relationship dialog, select 'has a variation named' from the Relationship Type drop-down box.

Enter a Descriptive Prefix of Manufactured.

Click the *OK* button.

3. The model diagram will be updated and the Product and Manufactured Product entities will appear something like this:



Note the following about the updated diagram:

- The Manufactured Product entity has inherited the identifying element of Product, Product Number, which is now the identifying element of Manufactured Product.
- The identifying element of Manufactured Product has a descriptive prefix of Manufactured.
- The descriptive prefix has been placed in the text of the relationship shape.
- Manufactured Product Number is denoted as being an inherited identifier by an asterisk in brackets (*) to its right.

• The full impact of the relationship on the two entities will be shown and discussed further in a later tutorial, where these entities will be built.

Step 3. Create a Supplied Product Entity and Relate it to the Product Entity

In this step, you will create another Variant entity, Supplied Product. A Variant entity is used to group elements which are part of the Product entity but specific to products which are provided from a supplier. The Supplied Product entity will be related to the Product entity.

- Create a Variant entity called Supplied Product.
 Set its physical file name to iiiPRS, where iii are your initials.
- Create an element for the entity called Supplier product.
 Select the Text data type.
 Set its field name to iiiPRDSUP, where iii are your initials.
- 3. Add a relationship to make Supplied Product a variant of Product. Set the Descriptive Prefix to Supplied.

Step 4. Create a Product Type Element for the Product Entity

In this step, you will create a Product Type element for the Product entity. In the resultant database, this element will be used to denote whether a Product is Manufactured or Supplied.

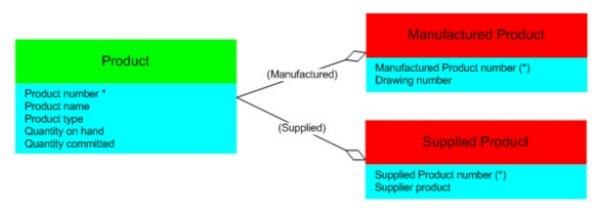
- 1. Open the Entity Elements window for the Product Entity.
- 2. Create an element called Product type.

Set its field type to Alphanumeric and its length to 10.

Set its field name to iiiPRDTYP, where iii are your initials.

- 3. Move the Product Type element immediately below the Product Name element.
- 4. Close the Entity Elements window.

The model diagram should now look something like this:



5. Save your model.

Summary

Important Observations

- A Variant entity can only inherit its identifying element(s) from another entity by means of a Variation relationship.
- You cannot assign an identifying element when working with elements of a Variant entity.

Tips and Techniques

- Ensure you create an element in the source entity of the Variation relationship that will hold the entity's type of variation in the resultant database.
- Variant entities and variation relationships are used when you want to avoid sparsely populated databases. For example, you could include all elements from the Supplied Product and Manufactured Product entities into the Product Entity. If you combine these entities into one file, you will notice that all fields relating to a Manufactured Product will be blank whenever a Product is a Supplied Product, and vice versa. Hence, you will have a database with many unused fields. Using variations allows information to be grouped or categorized. The Supplied Product file will only contains records for Supplied Products and not for every record in the Product File.

What I Should Know

- How to create a Variant entity.
- How to create a Variation relationship between a Data entity and a Variant entity.

LGM007 – User Views

Objective:

- To learn how to create a User View for an entity.
- Understand the impact of User Views when the database is built.

To achieve the tutorial objectives, you will complete the following steps:

- Step 1. Create a User View for the Product Entity
- Step 2. Create a User View for the Order Entity
- Summary

Before You Begin:

You may wish to review the following topics in the Logical Modeler Guide:

• User Views .

In order to complete this tutorial, you must have completed the following:

- LGM001 View a Model
- LGM002 Create an Entity
- LGM003 The Join Relationship
- LGM004 Conceptual Entities and the Includes Relationship
- LGM005 The Parent/Child Relationship
- LGM006 The Variant Entity and the Variation Relationship

Step 1. Create a User View for the Product Entity

In this step, you will create a user view for the Product Entity, sequenced by Product Name.

- 1. Ensure the model you created in the previous tutorial is open.
- 2. Select the Product Entity by clicking on its diagram shape.
- 3. Display the Maintain User Views window by selecting the *Edit* menu and choosing the *User Views*... option, or right-click the entity and select the *User Views*... option from the pop-up menu.

The Maintain User Views window is displayed:

Maintain User Views (Product)	
🤌 🖪 🗉 🗙	
<u> </u>	

A new user view can be created and existing user views can be changed and deleted.

4. To create a new user view, click the 🚨 New toolbar button.

The Specify User View window will be displayed:

Specify User View (Product)	
View name	
Logical view name	
Element	Кеу
C Product number	
Product name	
Product type	
Quantity on hand	
Quantity committed	
	Ŷ
	J
	OK Cancel

5. At the top of the window, a view name must be specified.

On the left pane of the window is a list of all elements within the entity. This includes elements that have been inherited by the entity as the result of relationships with other entities.

On the right pane of the window is a list that contains the elements that make up the view. These are known as the view's key.

- 6. Enter a View Name of By Product Name.
- 7. Enter a Logical View Name of iiiPRDNM, where iii are your initials.
- 8. Add the Product Name element to the view's key by double-clicking it in the Element list ,or by dragging it from the Element list to the Key list.

Note that when an element is added to the Key list, it is removed from the Element list. Double-clicking an element in the Key list or dragging it to the Element list will remove it from the Key list and add it back into the Element list.

Note that when the entity is built, the user view will be transformed into a logical file that implements the key that you have specified here.

9. Click the *OK* button.

The view name will be displayed in the list of user views.

- A user view can be changed by selecting it in the list and clicking the Change toolbar button.
- A user view can be deleted by selecting it in the list and clicking the Relete toolbar button.

10.Close the Maintain User Views window by clicking the 🎦 Close toolbar button.

Step 2. Create a User View for the Order Entity

In this step, you will create a user view for the Order Entity, sequenced by Order Status and Order Date.

- 1. Display the Maintain User Views window for the Order entity.
- 2. Create a User view called By Order Status, Order Date.
- 4. Enter a Logical View Name of iiiORDST.
- 5. Add Order Date to the key of the view.
- 6. Add Order Status to the key of the view.

Note that you have added the elements in the wrong order. You need Order Status to be the first element in the view's key.

- 7. Move Order Status above Order Date list by selecting either element and clicking the up or down arrow buttons to the right of Key list.
- 8. Click the *OK* button.
- 9. Close the Maintain User Views window by clicking the 🖻 Close toolbar button.

The full impact of the user views created in this tutorial will be shown and discussed further in a later tutorial, where the model will be built.

10. Save the model.

Summary

Important Observations

- Any element in an entity can be part of a user view, including those that are inherited as the result of relationships with other entities.
- Elements in the key of a user view can be re-sequenced.

Tips and Techniques

• Create user views for entities which have names or descriptions. Sequencing by name or description is a common requirement in many databases.

What I Should Know

• How to create a User View.

LGM008 – Abstractions

Objective:

- To learn how to create and use Abstractions.
- To understand how abstraction effect objects within a model.
- To create an abstraction which will contain all product related entities.
- To achieve the tutorial objectives, you will complete the following steps:
- Step 1. Create an Abstraction to Contain Product-Related Entities
- Step 2. Activate the Abstraction
- Step 3. Create a Supplier Entity and Relate it to the Supplied Product Entity
- Step 4. Deactivate the Abstraction
- Step 5. Drill down in Abstractions
- Summary

Before You Begin:

You may wish to review the following topics in the Logical Modeler Guide:

• Abstractions.

In order to complete this tutorial, you must have completed the following:

- LGM001 View a Model
- LGM002 Create an Entity
- LGM003 The Join Relationship
- LGM004 Conceptual Entities and the Includes Relationship
- LGM005 The Parent/Child Relationship
- LGM006 The Variant Entity and the Variation Relationship
- LGM007 User Views

Step 1. Create an Abstraction to Contain Product-Related Entities

In this step, you will create an abstraction to hold the Product, Manufactured Product and Supplied Product entities.

- 1. Ensure the model you created in the previous tutorial is open.
- 2. Drag the Abstraction shape from the Document Stencil onto the model diagram.

The Create Abstraction dialog will be displayed:

Create Abstr	action		
Name	[
		OK	Cancel

3. Give the Abstraction a name of Product Entities.

Click the *OK* button.

4. You now want to add objects to the Abstraction.

Select the *Abstraction* menu and choose the Specify Objects... option or right-click the Abstraction shape and select the Specify Objects... option.

The Specify Abstraction Objects window will be displayed:



A list of all objects is shown in the left pane. A list of entities within the abstraction is shown in the right pane.

5. Add the Product, Manufactured Product and Supplied Product entities to the Abstraction by double-clicking them or by dragging them to the Abstraction list.

As you add objects to the Abstraction, they are removed from the Object list.

To remove objects from an Abstraction, double-click them in the Abstraction list, or drag them from the Abstraction list to the Object list.

Note that you are referring to objects, as well as entities. Other Abstractions can be added to an abstraction.

Click the *OK* button.

6. The Abstraction shape on the diagram should appear as follows:



In addition to using the Specify Objects... option, you can add objects to an Abstraction using the following method:

- 7. Select the Abstraction by clicking on it.
- 8. Hold down the Shift key and select the objects you wish to add by clicking on them.
- 9. Release the Shift key.
- 10.Add the objects by selecting the Abstraction menu and choosing the Add *Objects* option, or by right-clicking the Abstraction and choosing the Add Objects.

Objects can be removed from an Abstraction by using the same selection method and the selecting the Remove Objects option.

Step 2. Activate the Abstraction

In this step, you will learn how to activate an Abstraction.

1. Activate the Abstraction by selecting it and then selecting the *Abstraction* menu and choosing the *Activate* option, or by right-clicking the Abstraction and choosing the *Activate* option.

The diagram will be updated to show only the Product, Manufactured Product and Supplied Product entities.

Note that the Product Entities Abstraction is still shown.

Note also that the title bar of the modeler has been updated to indicate that the abstraction is active.

2. Move the Product entity to a different place on the diagram. When the abstraction is activated/deactivated, you will see how the layout is saved.

Step 3. Create a Supplier Entity and Relate it to the Supplied Product Entity

In this step, you will create a Data entity called Supplier and relate it to the Supplied Product entity.

1. Create a Data entity called Supplier.

Set its physical file name to iiiSUP, where iii are your initials.

2. Create the following elements for the Supplier entity:

Element Name Data Type Field Name

Supplier number ID Number iiiSUPNUM

Supplier name Name iiiSUPNAM

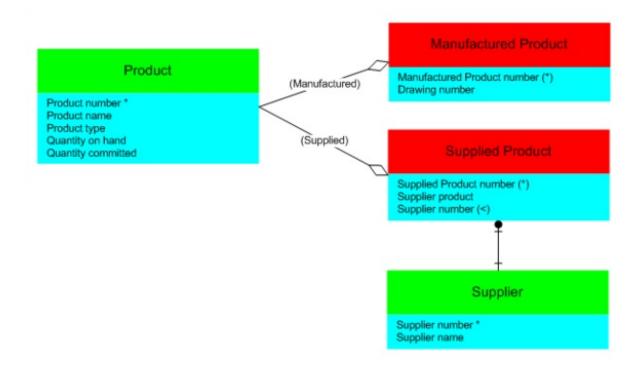
- 3. Set Supplier Number to be the identifying element.
- 4. Close the Entity Elements window.

Note that, because the Product Entities Abstraction is active, the Supplier Entity has been added to the Product Entities Abstraction automatically.

5. Create a Mandatory Join (must refer to) relationship between Supplied Product and Supplier by connecting a relate shape source to the Supplied Product entity and then target to the Supplier entity.

Leave its Descriptive Prefix blank.

6. The model diagram should now appear something like this:



Step 4. Deactivate the Abstraction

In this step, you will learn how to deactivate an active Abstraction. You will also add an Include relationship between Supplier and Address.

1. Deactivate the Abstraction by selecting it and then selecting the *Abstraction* menu and choosing the *Deactivate* option, or by right-clicking the Abstraction and choosing the Deactivate option.

The diagram will be updated to show all model objects. Note that the Product entity, which you moved in a previous step, moves back to its original position on the diagram. If you re-activate the abstraction, the Product entity will again move, to the position you specified when the abstraction was previously active.

Note also that the title bar of the modeler has been updated to indicate that the entire model is being shown.

You may need to reposition the newly added Supplier entity.

2. Create an Include relationship between Address and Supplier by connecting a relate shape source to the Supplier entity and then target to the Address entity.

The address elements are now included in Supplier.

3. Activate the Product abstraction again.

Notice that the Address entity is not shown in the abstraction even tough there is a relationship to this entity.

4. Deactivate the Product abstraction.

Step 5. Drill down in Abstractions

In this step, you will learn how you can use abstractions to drill down through a model and hide the complexity of a model.

1. Drag the Abstraction shape from the Document Stencil onto the model diagram.

The Create Abstraction dialog will be displayed:

Create Abstr	action		
Name			
		OK	Cancel

Give the Abstraction a name of My Model.

Click the *OK* button.

2. Activate the My Model Abstraction.

All entities in the diagram will disappear as no entities have been added to the abstraction. You have effectively hidden the complete model.

To drill down to show specific parts of the model, you can use your existing Abstraction.

3. Deactivate the My Model Abstraction.

The complete diagram is shown as no Abstractions are active.

- 4. Add Product Entities Abstraction to the My Model Abstraction.
- 5. Activate the My Model Abstraction.

The Product Entities Abstraction will be displayed.

6. Activate the Product Entities Abstraction.

The Product, Manufactured Product and Supplied Product Entities will now be displayed.

- 7. Deactivate the Product Entities Abstraction.
- 8. Deactivate the My Model Abstraction.
- 9. Save the model.

Summary

Important Observations

- Abstractions can be used to simplify your model by grouping related entities together under a common name.
- The positions of entities can be changed when the abstraction is active and will be remembered by the modeler whenever the abstraction is subsequently activated. Similarly, when the abstraction is deactivated, the entities move back to their original positions.
- The title bar of the Logical Modeler window tells you if an Abstraction is active or if the entire model is being shown.
- Creating objects while an abstraction is active will add those objects to the Abstraction automatically.
- All entities with relationships to objects in the abstraction are not shown. Changes to entities and relationships within the abstraction are still propagated to the entire model, however, and can affect entities that are not included in the abstraction.
- An Abstraction can be part of another abstraction in order to create a drilldown effect.

Tips and Techniques

- Use Abstractions if you have a large, complex model. When working only on a small part of the model, the appropriate abstraction can be activated to give an uncluttered view of the objects with which you wish to work.
- An abstraction is not the same as using the Pan and Zoom features when viewing a model. The Abstraction forms a group of entities as part of the model's definition.
- When working with entities in an active abstraction, you may change their positions. They will move back to their original positions when the abstraction is deactivated..
- An Abstraction cannot be deleted when it is active.
- Reminder: Do not create a loop in your abstractions when drilling down. For example, if Abstraction A contains Abstraction B, and Abstraction B contains Abstraction C, then do not include Abstractions A or B into C.

What I Should Know

- How to create an Abstraction.
- How to activate and deactivate an abstraction.

LGM009 – Build the Model

Objective:

- To learn how to built individual entities
- To learn how to build a model.
- To learn how to manipulate the Builder Queue.
- Understand what is built in the repository.

To achieve the tutorial objectives, you will complete the following steps:

- Step 1. Build the Address Entity
- Step 2. Build the Remainder of the Model
- Step 3. Correct Build Errors and Rebuild the Model
- Step 4. Review What Has Been Built in the Repository
- Summary

Before You Begin:

You may wish to review the following topics in the Logical Modeler Guide:

• Building a Model.

In order to complete this tutorial, you must have completed the following:

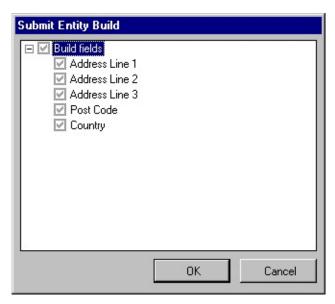
- LGM001 View a Model
- LGM002 Create an Entity
- LGM003 The Join Relationship
- LGM004 Conceptual Entities and the Includes Relationship
- LGM005 The Parent/Child Relationship
- LGM006 The Variant Entity and the Variation Relationship
- LGM007 User Views
- LGM008 Abstractions

Step 1. Build the Address Entity

In this step, you will build the Address entity. The Address is a conceptual entity so it will have no file definition.

- 1. Ensure the model you created in the previous tutorial is open.
- 2. Begin the build process for the Address entity by selecting Address and then selecting the *Build* menu and choosing the *Build Entity*... option, or by selecting Address and clicking the toolbar button, or by right-clicking Address and choosing the Build... option from the pop-up menu.

The Submit Entity Build dialog will be displayed:



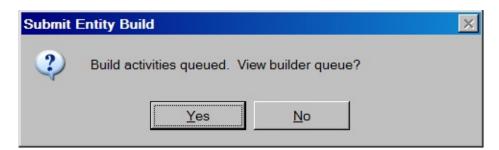
The dialog displays a list of build activities which will be performed for the entity.

Note that because the Address entity is a Conceptual entity, only its fields can be built.

Note also that, because neither the entity nor any of its elements have been previously built, all elements have been selected and cannot be deselected.

3. Click the *OK* button.

A message box will be displayed informing you that build activities have been queued and asking whether you wish to view the Builder Queue at this point.



4. Click the *Yes* button.

The following window will be displayed:

Build activity Build fields	Build status Ready to build
Build activity Build fields	Build status Ready to build
Build fields	Ready to build

This is the Builder Queue where repository builds are executed from.

The top pane of the window contains a list of build activities to be performed. Note that entities can have up to three build activities:

- Build repository fields,
- Build database file and
- Build relationships.

Each build activity has a status. By default, activities are submitted to the

queue in a Ready status. Individual entries can be deleted, held and released, in a similar fashion to an iSeries job queue.

5. Select the only entry in the list and click the 🔎 Hold toolbar button.

Note that the build activity status is set to Held. This means that the build activity will not be performed if you perform a build at this point – it must first be released.

6. Click the Selease toolbar button to release the build activity.

Note that its status is reset to Ready.

7. Click the 🧖 Build toolbar button to start the build.

A progress bar will be shown while the build is being performed. Note that the status of the build activity changes to Active and then to Built. A status of Built denotes that the build completed with no errors.

- 8. Click the 🙋 Close toolbar button to close the Builder Queue.
- 9. Display the Build Status view of the Address entity. It should appear something like this:



Note that the build status of the Address fields has been set to built and that the date and time of the status change is shown.

- 10. Redisplay the Elements view of the Address entity.
- 11. Display the Builder Queue by selecting the *Build* menu and choosing the *Builder Queue*... option or by clicking on the Build toolbar button.

The Build Repository Fields activity for Address is still in the queue with a status of Built.

12. Click the 🔽 Purge toolbar button.

The following window will be displayed:

Purge Build Queue	
T AI	
Ready to build	
🗹 Built	
Failed	
🗖 Held	
Purge	Cancel

This window allows you to purge the Builder Queue of unwanted entries. By default, entries with a status of Built or Failed can be purged. However, either All or any combination of the Ready, Built, Failed or Held statuses can be purged.

13. Click the *Purge* button.

Note that, because the entry in the list had a status of Built, it has been removed from the list.

14. Close the Builder Queue.

Step 2. Build the Remainder of the Model

In this step, you will learn how to build multiple model objects. All objects which have not yet been built will be queued in the order in which they need to be built, based on interdependencies with other entities.

1. Click the 🔛 Build Model button on the main toolbar.

A window will be displayed asking what type of build you wish to perform. Select Entities requiring build and click the OK button.

Build Model	
Build Type	
Entities requiring build	
O All entities	
OK	Cancel

Again, a message box will be displayed informing you that build activities have been queued and asking whether you wish to view the Builder Queue at this point.

2. Click the Yes button.

The following window will be displayed:

Builder Queue			- 🗆 ×
🖻 🛛 🗙 😖 👄 🛛	5 0		
Name	Build activity	Build status	-
Customer Product	Build fields Build fields Build fields	Ready to build Ready to build	
Supplier Order Ordered Product	Build fields Build fields	Ready to build Ready to build Ready to build	
Manufactured Product Supplied Product	Build fields Build fields	Ready to build Ready to build	
Customer Product	Build file Build file	Ready to build Ready to build	
Supplier Order Ordered Product	Build file Build file Build file	Ready to build Ready to build Ready to build	
Manufactured Product Supplied Product	Build file Build file	Ready to build Ready to build Ready to build	
Error Messages			_
Date Time	Message		
1			
1			

Note that all three types of build activity have been queued for all entities which have not yet been built. Note also that the activities are always queued in the same sequence:

- Build repository fields
- Build database file
- Build relationships
- 3. Click the Suild toolbar button to start the build.
- 4. A Build Options dialog will be shown, as follows:

Build Options			
Generate referential integri	ty validation rules)	
1	[OK	Cance

This dialog allows you to specify whether you want the modeler to generate referential integrity rules between related files when the Build Relationships activity is performed.

5. Leave the check box checked and press the OK button.

A progress bar will be shown while the build is being performed.

Note that the status of the build activity changes to Active and then to Built or Failed. The build will take approximately 1 minute.

Note that some of the build activities related to the Customer, Order and Ordered Product entities have failed.

6. Select the Build Database File activity for the Customer entity.

An error message explaining why this build activity failed is displayed in the Error Messages list at the bottom of the window.

The Physical File Name for the Customer entity was not specified, and so the database file cannot be created. (Remember, this is why the red triangle was displayed for the Customer entity.)

7. Now select the Build Relationships activity for the Customer entity.

An error message says that this activity has failed because the Database File has not been built for the Customer entity.

Similarly, the Build Relationships activity for the Order entity has failed for the same reason. The Customer entity has not been database built, and so a referential integrity rule cannot be created between Order and Customer.

8. Close the Builder Queue.

Step 3. Correct Build Errors and Rebuild the Model

In this step, you will correct the errors generated by the build and perform another build.

- 1. Display the Customer entity's Change window by selecting Customer and then selecting the *Edit* menu and choosing the *Change*... option, or by selecting Customer and clicking the
- 2. Enter a physical file name of iiiCUS for the entity, where iii are your initials.
- 3. Click the *OK* button.
- 4. Submit the model build by clicking the Build Model 🔛 Build Model button on the main toolbar.
- 5. Select the Entities requiring build option and click the OK button.
- 6. Click Yes to display the Builder Queue.
- 7. Purge the Builder Queue of all non-Ready entries. Note that the only entries left are related to the Customer and Order entities.
- 8. Click the Suild toolbar button to perform the build. Click the OK button on the Build Options dialog. All builds should be set to a status of Built once the build has finished.
- 9. Close the Builder Queue.
- 10.Use the Build Status view to review your model build status.
- 11.Save your model.
- 12.Exit the Logical Modeler.

Step 4. Review What Has Been Built in the Repository

In this step, you will review the repository objects that were built earlier in the tutorial.

1. Go to the Repository browser of the main window.

Find all files whose name start with your iii initials. Remember, the file name is based on the physical file names you entered for each entity.

There should be 7 files displayed: Customer, Ordered Product, Order, Product, Manufactured Product, Supplied Product and Supplier.

2. Select the Customer file and review the list of fields shown in the right-hand list.

Note the following about the list of fields:

- Customer Number is the primary key of the file.
- Two sets of address fields have been created as the result of the Includes relationships you defined for Delivery and Billing addresses.
- The field names for the address fields have the computer names you specified in the Entity Elements window. This ensured that the field names are unique within the file. Note that without specifying unique field names, it would be impossible to create two sets of addresses by using the Includes relationship.
- 3. Review the Access Routes for the Customer file.

Note the following:

- An Access Route has been created linking the Customer file with the Order file, as a result of the Join relationship you defined between the two entities.
- The File Accessed parameter of the Access Route is a logical view of the Order file, which was also created as a result of the Join relationship you defined.
- 4. Display the Rules and Triggers window for the Customer file.
- 5. Review the rules for the Customer Number field.

Note the following:

• A referential integrity rule has been created between the Customer and Order files that says that a Customer cannot be deleted if Orders exists for

it.

- Again, the actual file used by the rule is the logical file that was created as a result of the Join relationship between the two entities.
- 6. Review the rules for one of the Post Code fields.

Note that, because the Post Code field was defined by using a data type, the validation rules defined against that data type's field (STD_ZIP) have been inherited by Post Code. In this instance, the validation rule states that a value must be not be blank.

7. Find the iiiCUSNUM field in the Repository browser, where iii are your initials.

Review its help text.

Note that the notes you entered for the Customer Number element have become help text for the resultant field.

8. Select the Order file, iiiORDER.

Note that a Customer Number field has been created and assigned to the Order file as a result of the Join relationship with the Customer entity you created.

9. Review the Access Routes for the Order file.

Note that Access Routes have been created as a result of the Join with Customer and as a result of the Parent/Child relationship with Ordered Product.

10. Review the logical views for the Order file.

Note the following:

- A logical view has been created as the result of the Join relationship with Customer. It has the name iiiORDER01.
- A logical view has also been created from the User View you specified, sequenced by Order Status and Order Date. It has the name iiiORDST.

11. Select the Ordered Product file, iiiOPR.

Note that a logical view called iiiOPR01 has been created automatically by the modeler, and that it has Product Number as its sole key field. Remember, you did not specify this logical view name, but because Ordered Product's physical file name was only 6 characters in length, the modeler was able to use it, along with a sequential number, to create one for you. 12.Review other objects that have been created.

Pay close attention to:

- The help text of fields that has been created from the notes entered in the model.
- The primary keys of files, particularly those that have been influenced by Parent/Child relationships.
- Fields that have been inherited as the result of relationships with other entities. Note how the Prefix information of the relationship has been used.
- Access routes that have been created as the result of relationships.
- Logical files that have been created as the result of relationships and user views.

Summary

Important Observations

- Entities may be queued for build individually or the entire model can be queued for build. When building an entire model, the Logical Modeler analyses the build requirements and places build activities in the correct order, based on interdependencies with other entities.
- Build activities can be deleted, held and released once they are placed on the Builder Queue, in a similar fashion to an iSeries job queue.
- There are three phases of building: Repository Fields, Database File and Relationships. The Database File and Relationships phases are dependent on the successful completion of the build phases that precede them.
- Entity Build Statuses are updated with the date and time at which the three build phases were executed.
- Model objects are transformed into repository objects:
 - Elements are transformed into field definitions.
 - Element notes are transformed into field help text.
 - Conceptual entities are transformed into fields.
 - Data and Variant entities are transformed into database files.
 - Relationships between entities are transformed into inherited fields, access routes, logical files and referential integrity rules.
 - User views are transformed into logical files.

Tips and Techniques

- Use the Build Model function to perform builds. All entities which require building will be queued in the correct sequence based on interdependencies with other entities. This cannot be assured when building entities individually.
- Execute as many build cycles as you like and correct errors as you go. You do not need to concentrate on field and file names while constructing your model.

What I Should Know

- How to build an individual entity.
- How to build the entire model.

- How to use the Builder Queue.
- What is implemented in the repository when a build is performed.