Visible® Analyst Use Cases Tutorial

Visible Systems Corporation
24 School Street, 2nd floor
Boston, MA 02108
617-902-0767

www.visiblesystemscorp.com
https://twitter.com/VISIBLECorp

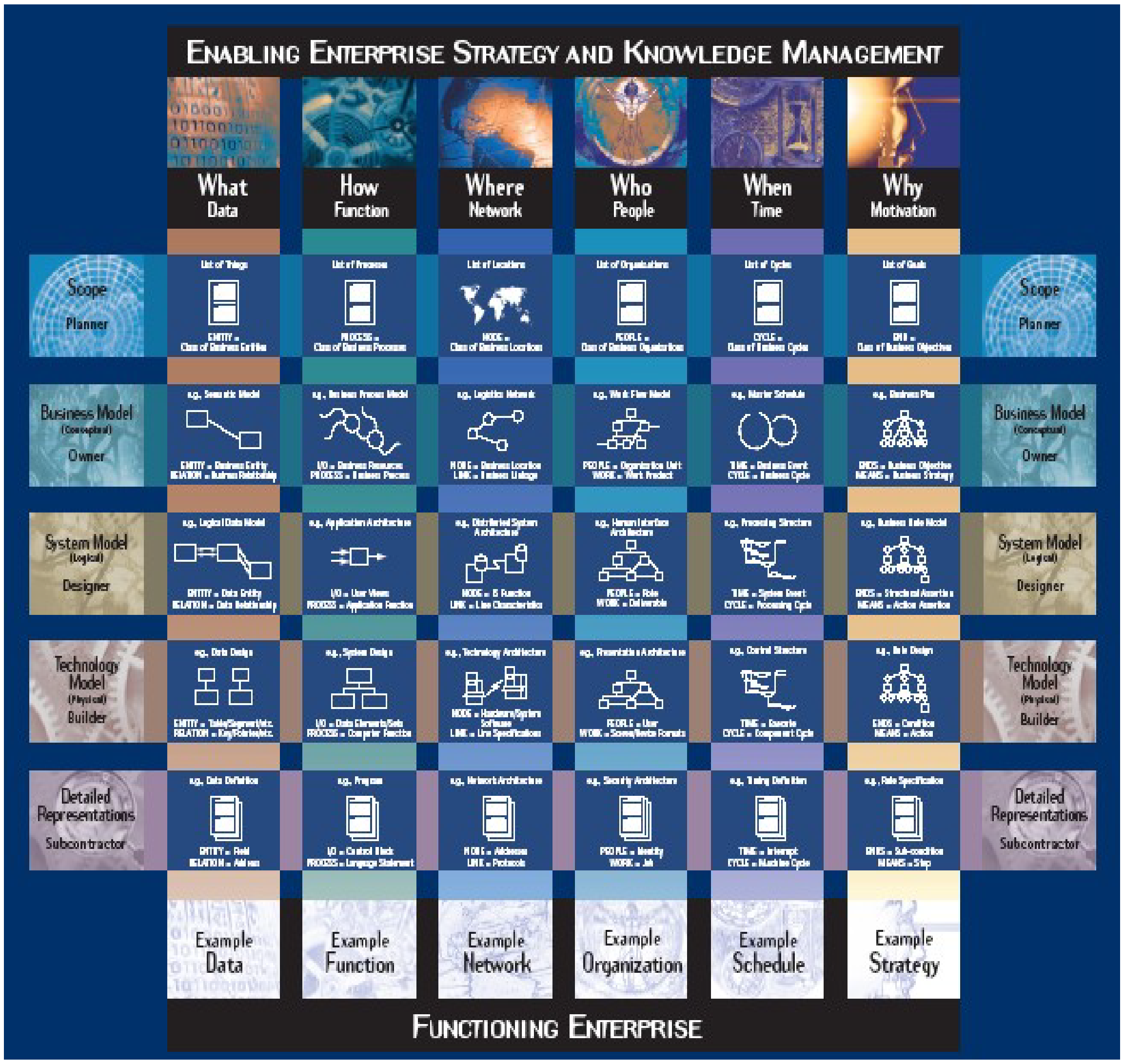
Email: contact@visiblesystemscorp.com

Visible®

Visualize. Align. Transform.™

Visualize patterns, align strategy and transform change into meaningful business outcomes.

Enterprise-wide Analysis, Design and Planning for Improvement.



This manual was prepared using Microsoft Word for Windows.

Visible Analyst
Tutorial on Structured Methods, Repository Management and The Zachman Framework

Dear Colleagues:

Thank you for your time in selecting our product, the Visible Analyst. At Visible, we take your time and effort seriously. To that end, we pride ourselves on delivering the most appropriate, value-oriented solutions. And, we feel that we offer the very best in product support that often differentiates us from our competitors.

As you read though the tutorial, please take the time to understand that our approach to software development is one of a model driven approach. Within the framework of this approach, Visible, in part, supports the Model Driven Architecture (MDA) as defined by the Object Management Group (OMG). This group, commonly referred to as the OMG, is an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise wide applications. For more information about the OMG and in particular their MDA specification, please reference their web site at http://www.omg.org/mda/.

In conjunction with a model driven approach, Visible has incorporated a framework to enable you to better plan and manage your Enterprise Architecture effort. In this edition, The Zachman Framework, is the framework of choice. However, you can customize the Visible Analyst to implement other frameworks like, for example, the US Federal Enterprise Architecture Framework (FEAF).

The Zachman Framework

INTRODUCTION

It has been Visible Systems Corporation's experience that no matter where you start in your application development activities, you will soon find yourself making certain ―assumptions‖ about things not under your control or outside of your scope. To confirm or validate these assumptions, you find yourself addressing the artifacts up and down the Zachman Framework rows and/or across the columns to capture the true drivers for the system: who? what? where? when? why? and how?[1] This means coordinating with the affected or interested business experts, system users, and management.

In 1987 John Zachman wrote, ―To keep the business from *dis*integrating, the concept of an information systems architecture is becoming less of an option and more of a necessity.[2]‖ From that assertion over a decade ago, the Zachman Framework for Enterprise Architecture has evolved and become the model around which major organizations view and communicate their enterprise information infrastructure. The Zachman Framework draws upon the discipline of classical architecture to establish a common vocabulary and set perspectives--a framework--for defining and describing today‗s complex enterprise systems. Enterprise Architecture provides the blueprint--or architecture--for the organization‗s information infrastructure and provides a framework for managing information complexity and managing change.

Today the Zachman Framework has become a standard for Enterprise Architecture used by many of the most successful organizations in the world. Evidence of the acceptance of the Framework has been apparent at the annual forums conducted by the Zachman Institute for Framework Advancement (ZIFA, www.zifa.com). At each forum, attendees hear presentations on the many different aspects and practical uses of the Framework. *Visible* fully supports both the concept and philosophy of the Zachman Framework. *Visible* helps clients gain greater control of their information systems and technology requirements through development of an enterprise-wide architecture.

[1] ―Visible and the Zachman Framework for Enterprise Architecture‖ by Alan Perkins p. 2. Copyright © 1997-2001, Visible Systems Corporation.

[2] ―A framework for information system architecture‖ by J.A. Zachman p. 454 IBM Systems Journal, Vol. 26, Nos. 3, 1987, ©1987, 1999 IBM.

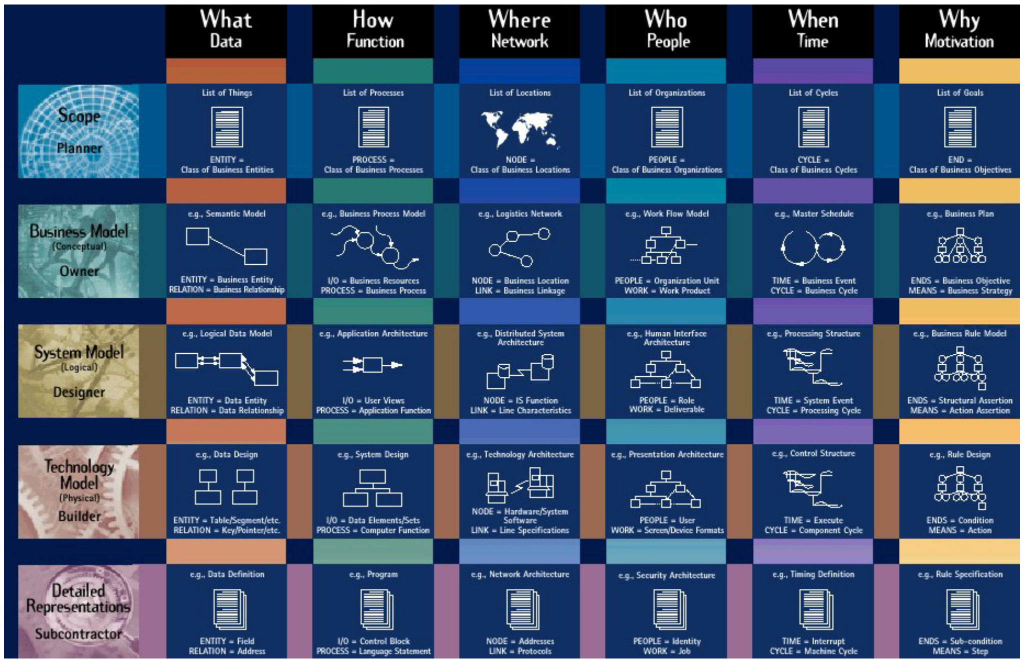
Visible takes an engineering approach to developing an enterprise architecture. We use a combination of forward and reverse engineering to establish the enterprise architecture. Forward engineering tasks include business planning and data and process modeling. Reverse engineering tasks include analysis and documentation of all existing structures for the organization. The result is a model that represents an integrated view of the enterprise architecture framework, with redundancies and discrepancies resolved and documented. All conceptual and logical architecture components can all be maintained in *Visible*'s proprietary modeling tool, *Visible Analyst®.*

The Visible Analyst supports the tasks and techniques involved in the creation and management of an enterprise architecture, with sufficient flexibility to integrate and support other approaches to software engineering. Visible Analyst captures business plans of multiple organization levels and maintains the hierarchy of planning components (mission, goals, strategies, measures, business rules, etc.).

Unlike many other modeling tools, Visible Analyst has the capability of directly linking each business plan component to the entities and attributes of a data model that support/implement the planning elements. This feature is used to control quality and completeness, and to ensure that process and system designs meet business requirements. Visible Analyst can also be used to specify physical information system designs based on the data model or import physical designs of existing data structures into the repository, and then link them back to the business plan component.

The following sections provide an overview of the Visible Analyst's repository and modeling capabilities, followed by an explanation of the Visible Analyst framework project. Each cell in the Zachman framework project is detailed in a cell-by-cell review including an explanation of the artifacts created for the cell in the Visible Analyst. A backup copy of the Zachman project is located in the VA\Zachman folder on the product CD. If you do not have a product CD, contact Visible Systems support at support@visiblesystemscorp.com for a copy of the document and backup file set. To access the project, see the Restore instructions at the beginning of this tutorial.

It is important to remember that the Visible Analyst Enterprise Project using the Zachman Framework is not a static one-time snapshot view of the enterprise. As mentioned in the cell explanations, the artifacts such as the business plan, physical data model, security architecture, strategic goals, etc. will change as the enterprise changes. Using the Visible Analyst and its repository to model the enterprise provides a one-stop location where all information about the enterprise is located. External documents may be changed, but the hyperlinks to the artifacts are maintained within the enterprise project, allowing for both a birds eye and physical implementation perspective of the enterprise.



	What Data	How Function	Where Network	Who People	When Time	Why Motivation
Scope Planner	List of Things ENTITY = Class of Business Entities	List of Processes PROCESS = Class of Business Processes	List of Locations NODE = Class of Business Locations	List of Organizations PEOPLE = Class of Business Organizations	List of Cycles CYCLE = Class of Business Cycles	List of Goals END = Class of Business Objectives
Business Model (Conceptual) Owner	e.g., Semantic Model ENTITY = Business Entity RELATION = Business Relationship	e.g., Business Process Model I/O = Business Resources PROCESS = Business Process	e.g., Logistics Network NODE = Business Location LINK = Business Linkage	e.g., Work Flow Model PEOPLE = Organization Unit WORK = Work Product	e.g., Master Schedule TIME = Business Event CYCLE = Business Cycle	e.g., Business Plan ENDS = Business Objective MEANS = Business Strategy
System Model (Logical) Designer	e.g., Logical Data Model ENTITY = Data Entity RELATION = Data Relationship	e.g., Application Architecture I/O = User Views PROCESS = Application Function	e.g., Distributed System Architecture NODE = IS Function LINK = Line Characteristics	e.g., Human Interface Architecture PEOPLE = Role WORK = Deliverable	e.g., Processing Structure TIME = System Event CYCLE = Processing Cycle	e.g., Business Rule Model ENDS = Structural Assertion MEANS = Action Assertion
Technology Model (Physical) Builder	e.g., Data Design ENTITY = Table/Segment/etc. RELATION = Key/Pointer/etc.	e.g., System Design I/O = Data Elements/Sets PROCESS = Computer Function	e.g., Technology Architecture NODE = Hardware/System Software LINK = Line Specifications	e.g., Presentation Architecture PEOPLE = User WORK = Screen/Device Formats	e.g., Control Structure TIME = Execute CYCLE = Component Cycle	e.g., Rule Design ENDS = Condition MEANS = Action
Detailed Representations Subcontractor	e.g., Data Definition ENTITY = Field RELATION = Address	e.g., Program I/O = Control Block PROCESS = Language Statement	e.g., Network Architecture NODE = Addresses LINK = Protocols	e.g., Security Architecture PEOPLE = Identity WORK = Job	e.g., Timing Definition TIME = Interrupt CYCLE = Machine Cycle	e.g., Rule Specification ENDS = Sub-condition MEANS = Step

Figure 2-1 Zachman Framework

Image provided courtesy of the Intervista Institute, Copyright © Intervista Institute (www.intervist-institute.com)

Business Planning Techniques

INTRODUCTION

Business rules are used to capture and implement precise business logic in processes, procedures, and systems (manual or automated). They can also provide the basis for expert systems. One way these business rules are captured in the Visible Analyst is through the creation of Strategic Planning statements, as well as the triggers, check constraints, and element definitions as explained below.

Enterprises that take a model driven approach to software component development can use business rules to refine the models and create better designs. An enterprise that properly documents its business rules can also manage change better than one that ignores its rules. Business rules may be any of the following:

- Definitions of business terms
- Data integrity constraints
- Mathematical and functional derivations
- Logical inferences
- Processing sequences
- Relationships among facts about the business

These types of business rules are examples of metadata. They can be defined as metadata, modeled as metadata, and, most importantly, they can be implemented as metadata for an enterprise's operational and strategic information management systems.

Implementing business rules as metadata is the most rigorous and, at the same time, most flexible approach to business rule implementation. This is in contrast to other implementation approaches.

- Process-driven approaches can be rigorous, but they are by no means flexible. In any process-centric approach, implementing the business rules is fairly straightforward, but, because they are typically implemented in code, changing them can be difficult and labor intensive.

- Procedure-driven approaches, characterized by manuals and checklists, are certainly flexible, but they are not rigorous. Procedures can be changed very easily. However, procedures are only as rigorous as their users choose them to be. People can and will ignore procedures.

VISIBLE BUSINESS RULES

Visible has pioneered a method and tools that allow business rules to be defined and implemented as metadata. Our approach captures rules as logical model elements and implements them as database tables, triggers, and object actions. The key to this is fully understanding the rules, documenting them consistently, and building conceptual information models and logical data models that accurately reflect the rules. *Visible*'s flagship modeling tool, ***Visible Analyst*®**, has the ability to store business rules in many formats in a single enterprise architecture model. This makes the rules *visible.* The rules can be modeled, including specific values for metadata. These migrate automatically to any database design generated from the models and, ultimately, are automatically inserted into the database tables when they are created or altered from the design.

This means that a business rule can be documented once in a logical model and still be part of multiple system designs and implemented databases. This one-logical, many-physical representation of business rules as metadata also allows a significant change in software component design. Components can be designed to access database tables for rules and do not have to include complex decision tables and rule-based processing logic.

There are several advantages that an enterprise can realize from using business rules as metadata:

- Allows maximum flexibility
- Reduces system maintenance
- Simplifies system design, development and implementation
- Rules can change without affecting implementation
- Ensures that systems fully support business needs
- MIS personnel don't need to learn the intricacies of the business

BUSINESS RULES IN BUSINESS MODELS

Business rules take several forms when documented in enterprise business models. There is one best representation that is appropriate for each particular type of business rule. For example, strategic business rules are best modeled as simple business statements, while operational business rules are best modeled as static data instances and derivations.

Business Statements

A business statement is a simple declaration. Business statements should be written in business language. They should be concise and clear. They should represent a single concept or idea. They should state business requirements, not system requirements. The kinds of strategic business rules that can be modeled using business statements are illustrated in Figure 3-1.

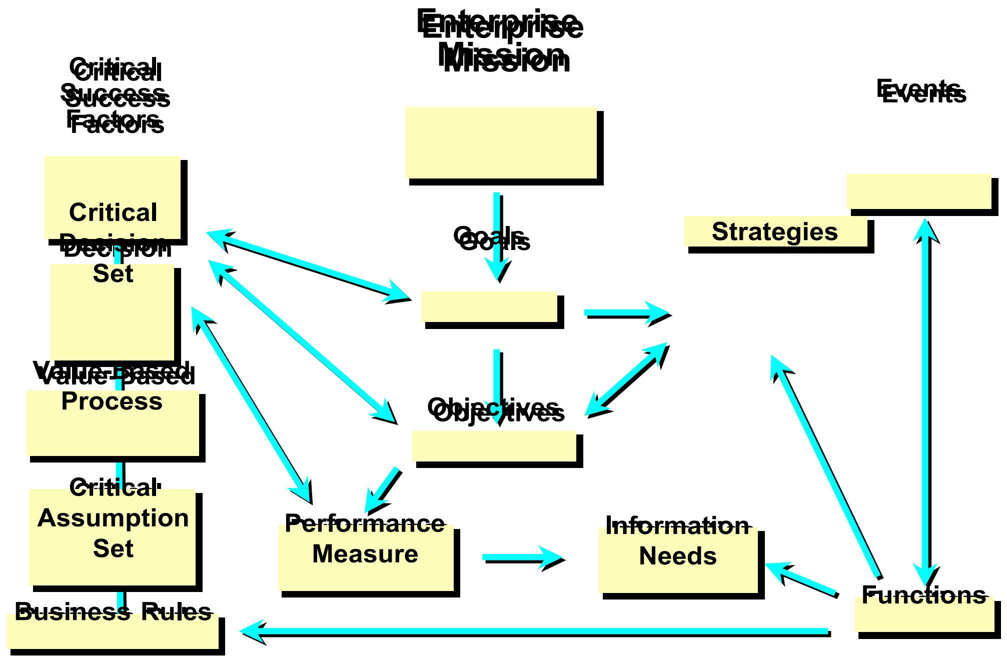


Figure 3-1 Business Rules

The Visible Analyst captures these business rules as Strategic Planning statements, which can be defined in the statement hierarchy then linked to other repository entries. This linkage provides both a validation of the rule as implemented by the object, such as a process, entity, element, check constraint, trigger, etc, and a graphical representation of the rules for communicating the business vision and rules that govern the organization. The hyperlink capability, available for any editable field in the Visible Analyst, can also be used to reference and link to external documents, web sites, applications, etc.

Use Case Diagramming

OVERVIEW

A Use Case diagram is a representation of a set of activities that are performed to satisfy user goals. It is based on the premise that a user interacts with a business system to obtain benefits and satisfaction.

A Use Case diagram defines a set of transactions and the flow of events that occur from the time the user first starts using the system until the user‘s goals are satisfied. The Use Case diagram includes _actors‘ representing the typical kinds of users that will interact with the system. The actors then communicate with _Use Cases‘, representing the activities that are needed to satisfy the user goals.

The Use Case diagram focuses on _what‘ a business process must do as opposed to _how‘ a business process is implemented.

DEFINITIONS

A Use Case diagram includes the following components:

System Boundary	A system boundary is a rectangular box representing the business processes supported by an information system.
Use Case	A Use Case is an elliptical shape representing an activity included in an information system.
Actor.	An actor is a stick person shape representing a role or a set of roles that a user plays.

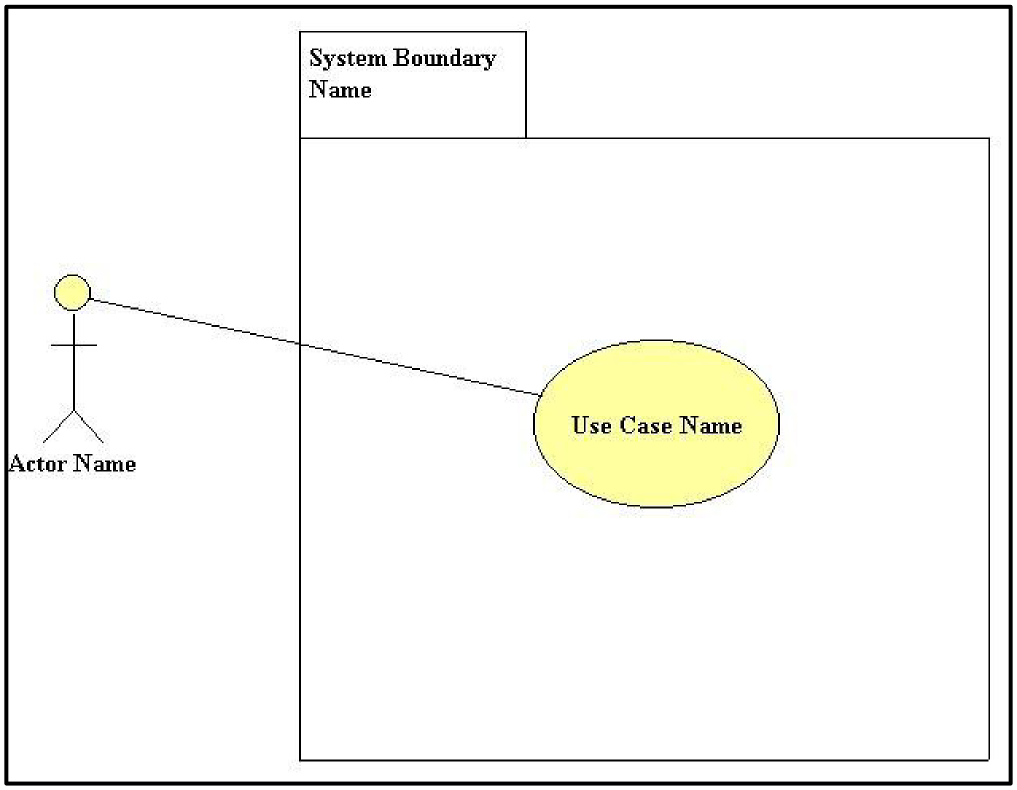


Figure 13- 1 Basic Use Case Components

The following general rules apply to the Use Case components:

- A Use Case symbol is positioned inside the system boundary.
- A Use Case diagram may include one or more Use Cases.
- An actor is positioned outside the system boundary.
- A Use Case diagram may include one or more actors.
- An actor communicates with one or more Use Cases.

RELATIONSHIPS

A relationship is represented as by a line from one object to another. The following relationships may be shown in a Use Case diagram:

- **Communicates**. The communicates relationship is represented by a solid line with no arrow, and is drawn between an actor and a Use Case.
- **Includes.** The includes relationship signifies that a Use Case contains behavior that is common to more than one Use Case. Thus the common Use Case is _included‘ in the other Use Cases. The includes relationship is represented with a dashed line with an arrow. An includes relationship may exist between Use Cases, but not between actors. The arrow points to the common or shared Use Case.
- **Extends.** The extends relationship signifies that one Use Case possesses behavior that enables the other (extended) Use Case to handle an exception, or a variation from the usual. The extends relationship is represented with a dashed line with an arrow to the basic Use Case. The extends relationship may only exist between Use Cases (not between actors). When specifying an extends relationship, you may also specify one or more extension points. An extension point represents additional information that needs to be gathered in order to complete the transaction.
- **Generalize.** The generalize relationship signifies that one thing is more usual or typical than the other thing. A generalize relationship may exist between two Use Cases, as well as between two actors. It is represented with a dashed line with an arrow. The arrow points to the Use Case (or actor) that generalize the other Use Case (or actor).

Examples of Relationships

The following is a discussion of the relationships presented in Figure 13-2.

- *Actor Communicates with Use Case*
  An example is an actor _communicates‘ with a Use Case. This is the most common relationship found on Use Case diagrams.
- *Use Case ‘Authorize Transaction’ included in Use Case*
  The Use Case _Deposit Cash‘ and _Pay Bill‘ include the common Use Case _Authorize Transaction‘.
- *Actor ‘Regular Customer’ Generalizes Actor ‘VIP Customer’*
  An example is an actor having a set of roles that _generalize‘ another actor‘s roles. For example, a _regular customer‘ is a generalization of a _VIP Customer‘.
- *Use Case ‘Arrange Financing’ Extends Use Case ‘Sell Automobile’*
  The extended Use Case provides additional steps concerning the setup of the loan. The basic Use Case _Sell Automobile‘ involves a cash sale.

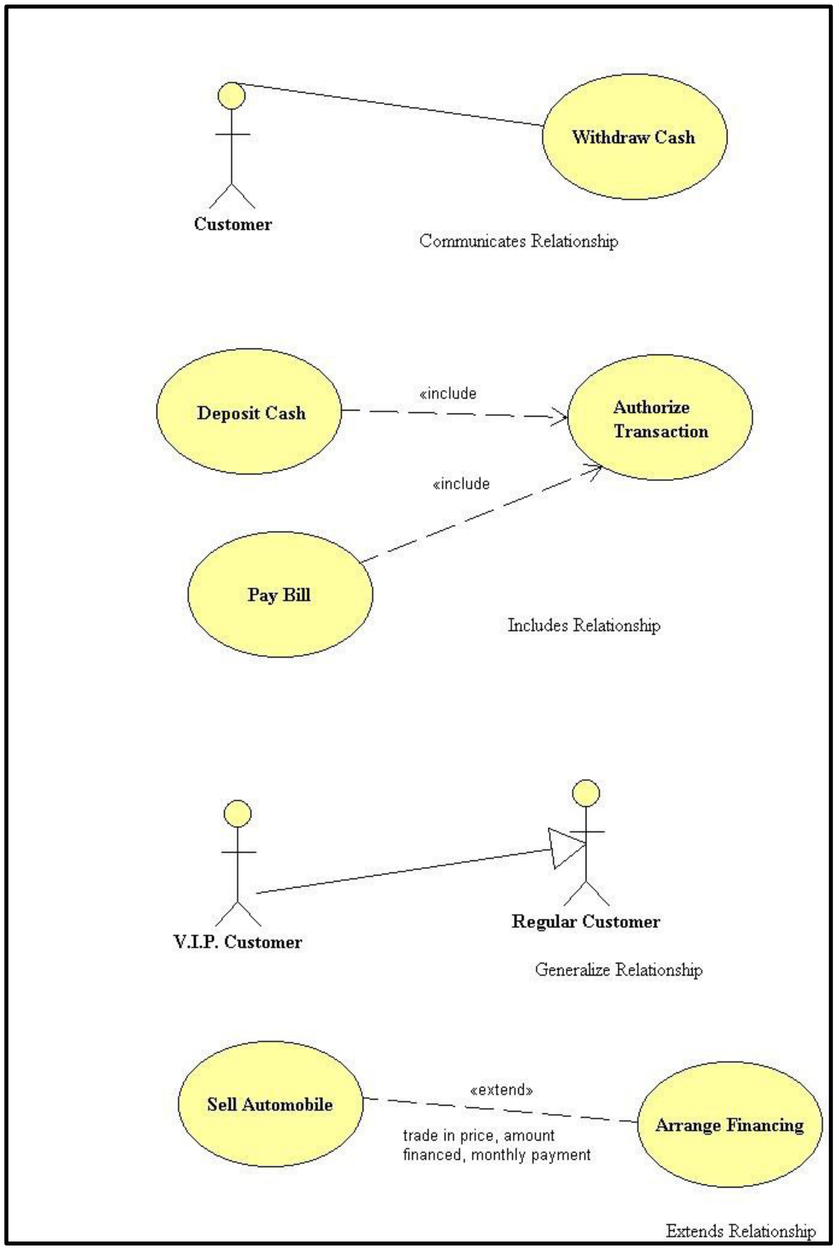


Figure 13-2 Use Case Relationships

DESCRIBING USE CASE ACTIVITIES

Visible Analyst provides the following basic attributes for describing a Use Case activity:

- **Name.** The name of the Use Case that appears on the diagram.
- **Description.** A brief description of the Use Case object.
- **Alias.** An alternate name for the Use Case. Up to ten aliases are permitted.
- **Scenario.** A complete description of the business scenario, up to 64,000 characters.
- **Notes.** Optional comments concerning the Use Case, up to 64,000 characters.

DEVELOPING YOUR USE CASE DIAGRAM

The business analyst creates one or more Use Case diagrams to fully explore the following questions:

Who will be the users of a business system?
Classify the kinds of users by defining an actor for each distinct role played by the users.

What goals does the system satisfy?
Consider the benefit or added value that the actor is seeking

What Use Cases are needed?
Consider the flow of events that will occur when a typical user who comes prepared with complete and accurate information and any prerequisites such as having enough cash or credentials. Make sure the Use Cases will satisfy the base case scenario. Then consider what activities are needed to handle the exceptions.

Ultimately, after exploring the above questions, you have one or more Use Case diagrams to create using Visible Analyst. Use the following procedures to create a Use Case diagram.

BUSINESS SCENARIO

The following is a business scenario that will be rendered using Use Cases.

An applicant visits a Department of Motor Vehicle‘s driver testing facility, and provides the registrar with a personal details including name, address, and telephone number, as well as proof of automobile insurance and a learner‘s permit. The registrar schedules a driver road test and written examination with a certified driver examiner. After passing the theoretical and practical tests the applicant is issued a driver’s license.

There are two actors in this scenario: the registrar and the driver examiner.

- The registrar interacts with the system by registering applicants, by setting up appointments for driver tests, and by issuing drivers licences to successful applicants.
- The driver examiners interact with the system by scheduling their availability, and by conducting driver tests.

The objective is to create a Use Case diagram that includes the following Use Cases:

- Register Applicant
- Schedule Driver Test
- Conduct Driver Tests
- Issue Licence

Adding System Boundaries, Actors, and Use Cases

The following procedure enables the analyst to create a Use Case diagram and add system boundaries, actors and Use Cases.

Set the Zoom Level:	1	From the View menu, select 66% zoom so you can see all the needed workspace.
Create a New Diagram:	2	From the File menu select New Diagram.
	3	Select the diagram type Use Case Diagram.
	4	Select Standard Workspace
	5	Click OK.
Add System Boundary:	6	Click on the third symbol button, the rectangle, on the control bar. This is a system boundary.
	7	Label this system boundary —Driver Registration System‖.
	8	Select the system boundary to highlight the system boundary, and its sizing handles. Then adjust the size and position of the system boundary so there is enough room to the left of the boundary to place actors, and enough room inside the boundary to place Use Cases.
Add Actors:	9	Click on the first symbol button, the actor, on the control bar. Place the symbol on the diagram.
	10	Label this actor —Registrar‖.

	11	Add the rest of the actors as shown in Figure 13-3.
Add Use Cases:	12	Click on the third symbol button, the Use Case, on the control bar. Place the symbol on the diagram.
	13	Label the Use Case —Register Applicant‖.
	14	Add the rest of the Use Cases as shown in Figure 13-3.
Save the Diagram	15	Save the Use Case diagram with the label Introductory Use Case.

Adding Relationships

The following procedure enables the analyst to add relationships to a Use Case diagram.

Adding a Communicates Relationship between an Actor and Use Case:	1	Select the first relationship button, the solid line Communicates. Begin the line on the actor —registrar‖ and end the line on the Use Case —Register Applicant‖.
Adding an Includes Relationship between Use Cases:	2	Select the second relationship button, the dotted arrow Includes. Begin the line on the Use Case —Register Applicant‖ and end the line on the Use Case —Collect Payment‖.
	3	Add the rest of the relationships as shown in Figure 13-3.
Save the Diagram:	4	Save the Use Case diagram.

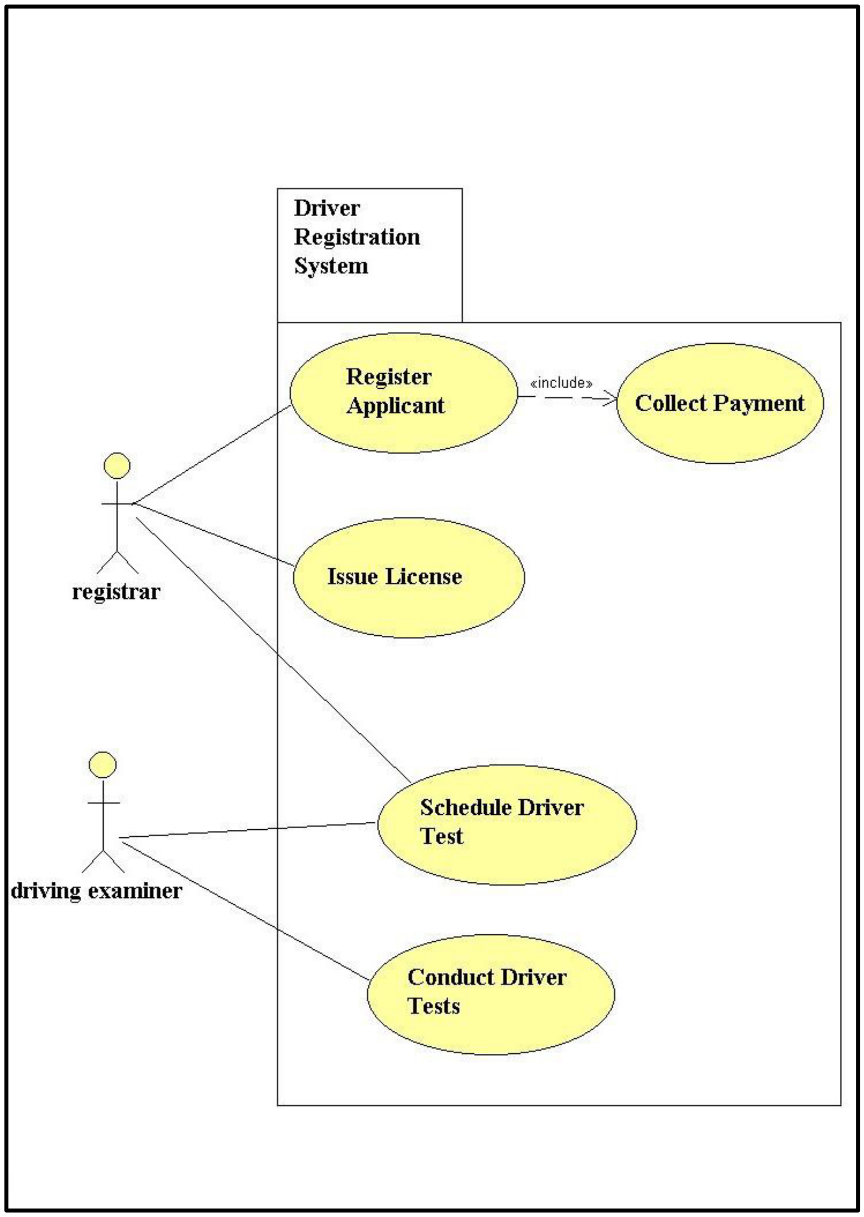


Figure 13-3 Introductory Use Case Diagram

Working with the Repository Functions

OVERVIEW

This unit helps familiarize you with the operation of the Visible Analyst repository and shows you the power of an online, interactive database for systems analysis, design and data modeling. The TEST project used in the previous lessons is used as the basis for your exercises.

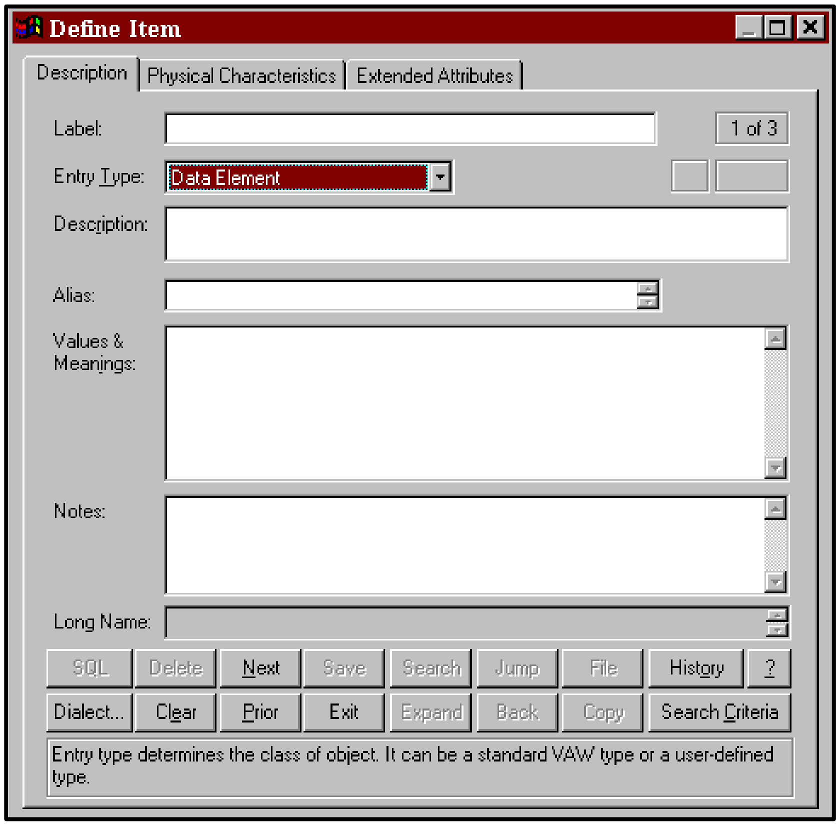
The repository is a powerful tool for creating and managing the narrative portions of a system‘s specification. A project repository is used to provide an entry location for all project documentation. Each graphical entry on your diagrams has an automatically created corresponding entry in the project repository, as do any items entered into a Composition or Alias field.

You have the ability to thoroughly define all of your graphical entries in the repository or to simply enter notes about them in the Notes field. As an integrated part of Visible Analyst, the repository operates in parallel with the diagramming functions to accomplish data decomposition logically. It contains powerful data management, text editing, import/export, and report facilities. By using it, meaning can be ascribed to diagrams and an asset of ever-increasing value can be created. After defining items, changing entries and entering notes, you can generate reports from this information in many different forms.

When you finish defining your data and processing, the repository also allows you to put it into an ASCII file and export it. The ASCII file can then be sorted to move data specifications to your database and process specifications to your text editor for writing code. (The Shell Code Generation utility can also be used for this purpose.)

Note

🗎 Users of the Educational and Demonstration versions of Visible Analyst cannot add items directly into the repository. First add the object to the diagram, and then edit it into the repository.



Entry type determines the class of object. It can be a standard VAW type or a user-defined type.

Figure 17-1 Blank Repository Dialog Box, Page One

REPOSITORY BASICS

Repository Control Buttons

The repository control buttons (see Figure 17-2) are always displayed at the bottom of the repository dialog box. Each of the button functions is accessed by clicking on the button or, as is customary in Windows, by using its keyboard shortcut by holding down the ALT key and pressing the underlined letter to execute the button function. Only the functions available to you at a given time are active; the others are grayed. The button functions are:

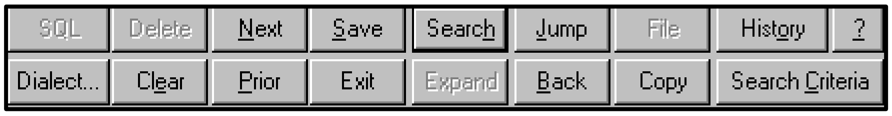


Figure 17-2 Repository Dialog Box Control Buttons

SQL	This button opens the Generated SQL for View dialog box. This dialog box displays the SQL generated for the current view object based on the view table and column specifications selected when creating the view, as well as the current SQL dialect. This button is active only when the entry type is View.
Dialect	This activates the RDBMS SQL Dialect dialog box. From there, you can change the current SQL dialect.
Delete	This deletes the current repository entry from the database. An entry can only be deleted when it has no location references, meaning that it does not appear on a diagram nor as an attribute of another repository item.
Clear	This clears the display of an entry and displays a blank repository dialog box. This allows you to Search for an existing entry or add a new entry. If you have made changes, you are prompted to save them before clearing. Your current location in the repository remains unchanged.
Next	This displays the next sequential repository entry that meets the repository search criteria (see below).
Prior	This displays the previous sequential repository entry that meets the repository search criteria.
Save	This saves all changes made to an entry.
Exit	This or the ESC key exits from the repository.
Search	This initiates a search for a particular entry in the repository. The procedure is explained in the section on Search Capabilities.
Expand *Contract*	This allows you to expand or contract the display size of some fields. The fields that normally display four lines can expand to display 15.

Jump	This allows you to jump immediately to another entry that is referred to in the current one. This feature is described in the section on Navigation Capabilities.
Back	This button provides a means of jumping to the previous repository entry. You can then continue to move backwards displaying previous repository entries.
File	This allows you to insert text from a DOS file at the cursor position or to copy highlighted text to a DOS file. It is explained in further detail in the Visible Analyst *Operation Manual* and in the online help system.
Copy	This button provides a means of copying the current object.
History	This provides a means of jumping back to a previously displayed repository object. A list is kept of every object definition that has been displayed. If you choose this button, the History dialog box appears and you can jump between entries by double-clicking on an entry. The maximum is 500 objects.
Help(?)	This displays context sensitive help about the repository. You can also press F1 to activate the help system.
Search Criteria	This allows you to specify how the repository is to be searched. It is explained in the section on Search Capabilities.

Other buttons that may be displayed on the Define dialog box are:

Primary Key	If the current object being examined is an entity type, the primary key button is displayed to the left of the Composition/Attributes field.
Attributes Details	This button provides a means of populating the composition of a repository entry with components and physical information. This button is displayed to the left of the Composition/Attributes field.

When the Entry Type is a Class, or when the Classic User Interface is turned off, the Add button displayed beneath the Attributes Details button is active. You can use this button to add details. When you begin typing in the field next to the Add button, the button is enabled. Click the Add button to add the attributes to the Attributes field.

Editing Keys

Because the Edit menu is not accessible from the repository, you can use the right-click menu that is available when an object (text) is highlighted and you click the right mouse button. Using the right-click menu, you can Cut, Copy, Paste, or Delete the selected object.

Field Types

The data repository of a Visible Analyst project is displayed using Define dialog box variations corresponding to different diagram objects. You see and work with some of these variations during the course of this lesson. The basic dialog box, shown in Figure 17-1, is for data elements, aliases, miscellaneous objects and external entities or source/sinks. Other objects, such as data stores, processes, functions, entities, relationships, modules, data flows information clusters, etc., have variations in individual fields and tabs of the Repository dialog box to accommodate the specific needs of those items. Some of these differences are seen later in the lesson.

Label Field

This is the name of the repository item. The names of items drawn on diagrams are automatically entered here.

Entry Type Field

This tells Visible Analyst what kind of object the item is: process, data flow, entity, etc. The entry type can be entered manually, or you can select the type from the scroll box accessed by clicking the down arrow at the end of the entry type field.

Note

🗎 You can edit the Entry Type and Label fields of data elements and data structures that do not appear on diagrams. The entry type for a data element cannot be changed if physical information for that element has been entered.

Description Field

The Description field is a two-line field that provides a convenient place to enter a somewhat more extensive descriptive title of the object than the Label field allows. The contents of this field are used for the Comment on Column (data elements) and Comment on Table (entities) when SQL DDL is generated if the selected SQL dialect supports this syntax.

Alias Field

The Alias field contains 10 lines of 128 characters each. It allows for the entry of alternative labels to the one used as the object label. This is most commonly used for indicating the cryptic abbreviations that are sometimes used in the actual coding of a software program, as opposed to the plain English names that are desirable for reference. The Alias field is an intelligent field. Data names entered into it establish new repository entries for these aliases.

Attributes Field

The purpose of the Attributes field is to accumulate the collection of data elements that you wish to define as constituting a data flow, entity, data store, etc. The Attributes field is an intelligent field. Data names entered into it establish new data element repository entries or update existing ones. These new data elements can then be used for further definition. Data flows, data structures and couples can also appear in some Attributes fields.

When you click the Attributes Details button, the Add Attributes dialog box appears. Using this dialog box, you can define up to 12 components and some of their properties. As you enter items, the dialog box automatically scrolls as necessary to allow you to enter more items until you reach 12. When you complete the entries, click OK to add them to the Attributes field. If you need to add more than 12 components, click the Attributes Details button again; and a new dialog box opens so that you can add additional attributes.

Use the Add button at the bottom of the Attributes field to add components one at a time. When you begin typing in the field next to it, the Add button becomes active. Complete your entry, and then click Add to enter the component in the Attributes field.

Values & Meanings Field

The Values & Meanings field allows an unlimited number of lines. The maximum number of characters that can be contained in the field is 64K. This field allows the entry of specific information about the value(s) the item can take.

Discriminator Values & Meanings Field

If the current object is a data element that is used as a discriminator, this field contains a list of values to identify the subtype entities. For each subtype, a value can be entered that will uniquely identify it. By default, these values are numbers starting with 0 for the supertype. To change the value, click the value until an edit control appears, make your changes, then press ENTER.

Notes Field

The Notes field is also a field that allows you to enter any pertinent information about the object. The maximum number of characters that can be contained in the field is 64K. This is the logical field to use when creating hyperlinks to external documents, web pages or other application files.

Location Field

This field displays two types of usage information. The field can contain the diagram name (and, for DFDs, the diagram number) of every diagram where the item appears. The field can also tell you if the item appears as an attribute of another item. This second kind of location entry has the entry type of the parent item, followed by an arrow and the name of the parent item.

Other Pages and Fields

Other pages of the Define dialog box contain additional information. For example, pages 2 and 3 of the basic repository form provide location and relationship information and specifications for PowerBuilder/VISION extended attributes. These two pages are similar for most entry types. For some entry types, additional pages will be displayed:

- When the entry type is an entity, the next five pages contain keys, foreign keys, triggers, check constraints, and physical information.
- For views, the next five pages provide table, column, join, clause, and option information.
- When the entry type is a relationship, there are additional pages that contain foreign key and cardinality information.
- When the entry type is a tablespace, an additional page contains property information.

A full list and complete descriptions of pages and fields can be found in the *Operation Manual* and in the online help.

Object Repository

The Visible Analyst repository provides several additional forms and data input components for supporting the object-oriented concepts. The object repository components are detailed below.

Attributes

The Attributes field replaces the Values & Meanings field whenever the Repository dialog box displays a class. The field contains a list of the data members for the class showing the local data element and type. To add, change, or remove local data elements, click the Attributes Details button or select Add/Change from the Repository Object menu. For each attribute, the following information can be defined:

- **Name**. The name of the attribute. Each attribute of a class has a separate entry in the repository with a type of local data element. This is an optional field. The search button can be used to find other local data elements in the repository.
- **Type**. The attribute type can be a class, data element, or data structure. If the type does not exist in the repository, a new class is created. The location field of the attribute type contains a reference to the current class. This is a mandatory field. The Search button can be used to display a list of valid types. If the attribute type is a data element or elemental

class, its physical characteristics are displayed. Entries added to the Type field are saved as data elements for an entity or data flow, and class/subtype element when the object is a class.

- **Limit**. The number of occurrences of the attribute. If this field is blank, the attribute occurs once.
- **Reference**. A qualifier to indicate the access method for an attribute. *Value* indicates the object defined in the *Type* field is used; *Address* indicates a pointer to the object is to be used; and *Reference* indicates a reference to the object is to be used. The default is Value.
- **Visibility.** *Publi*c members have global visibility. *Private* members are only accessible to member functions and friends. *Protected* members are accessible to derived classes and friends. *Implementation* members are only accessible to the class itself. The default is Private.
- **Qualification**. *Constant* indicates a member's value cannot be changed. *Volatile* indicates the member can be modified by something other than the program, either the operating system or hardware. *Static* indicates there is only one instance of the member regardless of the number of times a class is instantiated. The default is None.
- **Physical Characteristics**. If the attribute type is elemental, the physical characteristics can be set.

For every item entered into the Type field, Visible Analyst creates a repository entry (if one with the same name does not already exist) and updates that entry's location field. If an item is removed, this field is updated to reflect this. These repository entries are generally created as classes unless a data element already exists with the same name or the physical characteristics are defined

As you enter items, the dialog box automatically scrolls as necessary to allow you to enter more items until you have finished. Insert is used to insert a new attribute into the list at the current position, while Delete removes the current attribute (the current position is indicated by ➢➢). When you have completed the entries, click OK to add them to the Attributes field.

Item names entered into this field may contain up to 128 characters each and may consist of any upper or lower case letters, numbers, spaces, periods, underscore characters and hyphens; but the first character must always be a letter.

Attached Entities/Classes

The attached entities/classes for the currently displayed relationship are listed in this field. When an inheritance relationship is displayed, the characteristics of that relationship can be changed (see changing Inheritance Characteristics later in this chapter). Otherwise, the information cannot be edited from within the repository; and all changes must be made on a diagram. The field lists the two entities or classes attached to this relationship. Below the second entity name is listed the reverse of the current relationship. If either direction of the relationship has not been named, the name of the relationship in the reverse direction is displayed as ―reverse of (opposite relationship name).‖ This field allows you to jump to the repository entries for any of these entities or relationships, as described above.

Relations

For an entity or class, the Relations field displays the relationship name followed by the name of the entity or class on the other end of this relationship for each relationship attached to this entry. These sets are ordered alphabetically by the opposite entry name. When an inheritance relationship is displayed, the characteristics of that relationship can be changed (see Changing Inheritance Characteristics later in this chapter); otherwise, the information cannot be edited from within the repository; and all changes must be made on a diagram.

This field allows you to jump to the repository entries for any of these entities, classes, or relationships by positioning the cursor on the line containing an entity, class, or relationship name and clicking the Jump button.

Long Name

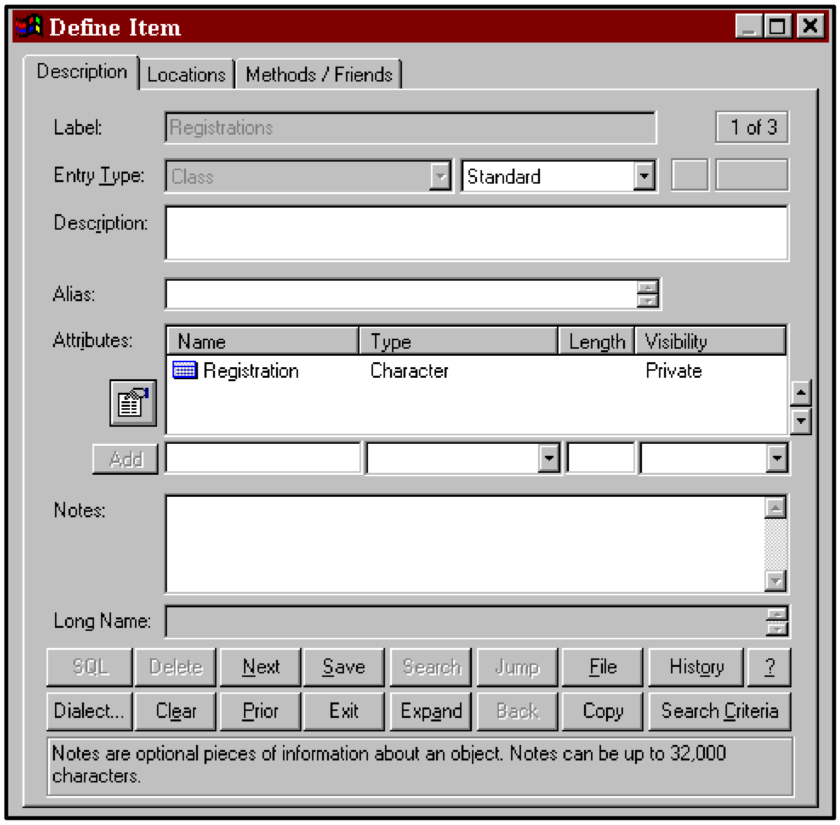
When a repository entry, either a local data element or a module, belongs to a class, the full name of the entry includes the class name. The Long Name field displays this name and, in the case of modules, includes the argument list (the argument list is required to differentiate overloaded member functions). If you want to change the argument list for a class method, click the right mouse button on the Long Name field and select Change (see the Methods section later in this chapter for details). If you want to change the class to which the method belongs, select Class from the Repository Object menu. To display the class definition, click the Jump button.

Class Characteristics

Concurrency, displayed on the Methods/Friends tab, is a class property that distinguishes an active object from inactive object. An *active* object may represent a separate thread of control. A *sequential* object is a passive object whose semantics are guaranteed only in the presence of a single thread of control. A *guarded* object is a passive object whose semantics are guaranteed in the presence of multiple threads of control.

A persistent class exists beyond the lifetime of an executable program. This means it must be stored on a non-transitory storage device. If the subtype of a class is set to either entity (associative or attributive) and the class is used on an entity relationship diagram, this field cannot be changed.

An abstract (or virtual) class cannot be instantiated because it contains pure virtual methods. If pure virtual methods exist for a class, Abstract is checked. If you attempt to uncheck this field, all pure virtual methods are reset to virtual. If you attempt to check it and virtual methods exist, they are converted to pure virtual methods.



Label: Registrations — 1 of 3

Entry Type: Class — Standard

Description:

Alias:

Attributes:

Name	Type	Length	Visibility
Registration	Character		Private

Notes:

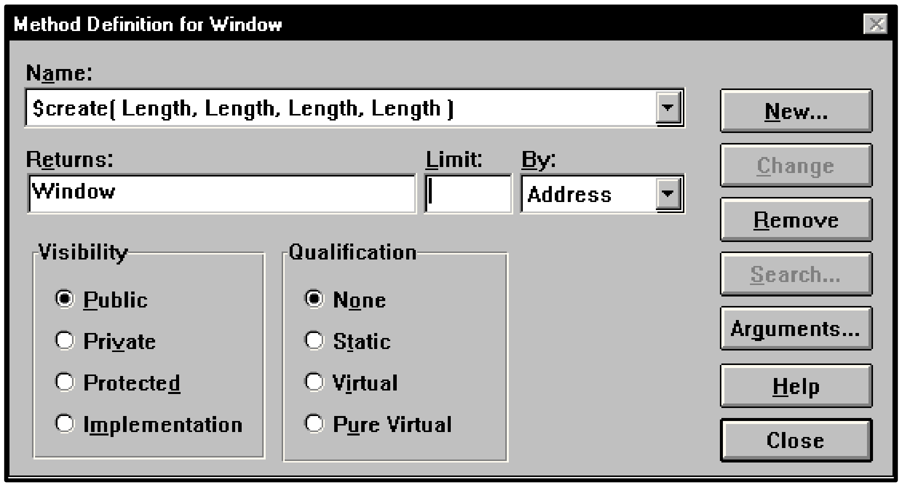
Long Name:

Notes are optional pieces of information about an object. Notes can be up to 32,000 characters.

Figure 17-3 Class Attributes

Methods

Methods (or Member Functions) are the operations that are defined for accessing a class. The Methods field contains a list of the functions for a class showing the name, return value, argument list, and flags to indicate its visibility. To add, change, or remove methods, click on the Methods field and click the Attributes Details button or select Add/Change from the Repository Object menu. To add a new method for a class, click the New button and type the name of method you wish to add. To search for methods that have already been defined in the repository, click the Search button. The list contains all modules that have previously been defined in the repository. If the module already belongs to a class, the class name is displayed. Note that when you select a module that already exists, the complete definition for that module is used including return value and argument list. Click OK to add the method name to the list of methods for the current class. For each method, the following information can be defined:



Name: $create(Length, Length, Length, Length)

Returns: Window

Limit:

By: Address

Visibility: [✓ Public] Private Protected Implementation

Qualification: [✓ None] Static Virtual Pure Virtual

Figure 17-4 Class Methods

- **Returns**. The return type can be a class or data element. If the type does not exist in the repository, a new class is created. The Location field of the attribute type contains a reference to the method. This is an optional field. Click the Search button to display a list of valid types.
- **Limit**. The number or size of the parameter. If this field is blank, it occurs once.
- **By.** A qualifier to indicate how the return value is passed. *Value* indicates a copy of the parameter is passed; *Address* indicates a pointer to the object is to be used; and *Reference* indicates a reference to an object is to be used.
- **Visibility**. *Public* methods have global visibility. *Private* methods are only accessible to other member functions within the same class and friends. *Protected* methods are accessible to derived classes and friends. *Implementation* methods are only accessible to the class itself. The default is Public.
- **Qualification**. *Static* indicates a method can be used without a specific instance of an object (it can only be used with static attributes (data members)). A *Virtual* method is one that you expect to be redefined in a derived class. A *Pure Virtual* method has no definition and must be defined in a derived class. A class with any pure virtual functions is an abstract (or virtual) class. The default is None.
- **Arguments**. A list of parameters to be used by the method. This is an optional field. If a method appears more than once with the same name within a class, it must have a different argument list for each definition. This is known as function overloading. See the next section for defining arguments.

When a method is added to a class definition, an entry of type module is created in the repository. The long name includes the class name and the argument list. The argument list is needed to differentiate between overloaded functions.

Note

🗎 Because the same name can be used for more than one method, there may be duplicate module entries in the repository, each belonging to a different class.

Arguments for Methods

When defining methods (member functions) for a class, the parameters to the function need to be specified. To add, change, or remove arguments, click the Arguments button on the Method Definition dialog box. For each argument, the following can be defined:

- **Name**. The name of the parameter. This is an optional field.
- **Type**. The parameter type can be a class or data element. If the type does not exist in the repository, a new class is created. This is a mandatory field. The Search button can be used to display a list of valid types. If the parameter type is a data element or elemental class, its physical characteristics are displayed.
- **Limit**. The number or size of the parameter. If this field is blank, it occurs once.
- **Pass By**. A qualifier to indicate the how the parameter is passed. *Value* indicates a copy of the parameter is passed; *Address* indicates a pointer to the object is to be used; and *Reference* indicates a reference to an object is to be used. The default is Value.
- **Qualification**. *Constant* indicates a parameter‘s value cannot be changed. *Volatile* indicates the parameter can be modified by something other than the program, either the operating system or hardware. The default is None.
- **Physical Characteristics**. If the parameter type is elemental, the physical characteristics can be set.

For every item entered into the Type field, Visible Analyst creates a repository entry (if one with the same name does not already exist). These repository entries are generally created as classes unless a data element already exists with the same name or the physical characteristics are defined.

As you enter items, the dialog box automatically scrolls as necessary to allow you to enter more items until you have finished. INSERT is used to insert a new parameter into the list at the current position, while the DELETE key removes the current parameter (the current position is indicated by ➢➢). When you have completed the entries, click OK to update the method name field. Item names entered into this field may contain up to 128 characters each and may consist of any upper or lower case letters, numbers, spaces, periods, underscore characters and hyphens; but the first character must always be a letter.

Friends

The Friends field displays a list of both friend classes and methods (or functions). A friend is allowed access to the private data members of a class. To add friends, click on the Friends field and click the Search button, select Add from the Repository Object menu, or double-click on the Friends field while pressing CTRL key. A list of classes and member functions is displayed in the Search list box. Locate each repository item you want to place in the Friends field and click the Search button; the item is added to the Select list box at the bottom. When you have found all of the entries you want, click the Select button and they are entered into the Friends field.

To remove a friend, highlight the desired item and press the DELETE key or select Cut or Delete from the Repository Object menu.

Navigation Capabilities

In this section, you change the displayed repository entry using Next, Prior, and Jump.

Note

The repository saves some internal settings for the duration of a Visible Analyst session. If these are set incorrectly, they may interfere with the smooth flow of this lesson. Therefore, we suggest that if you or another user worked in the repository during the current session, you should exit to Windows and restart Visible Analyst. In this way, you have a clean slate on which to run this lesson.

Open the Repository: 1 Access the repository using either Define from the Repository menu or CTRL+D. A blank Define dialog box is displayed.

Access an Entry: 2 Type ―Person Informationǁ in the Label field and press ENTER twice. (Pressing ENTER once brings up the Search dialog box. Pressing it a second time displays the entry found. If you press ENTER twice quickly, you get the same result without displaying the search box.) The repository entry for Person Information displays with all of the information that has been entered into the repository for this entry.

Move Around: 3 Click Next. The next entry in alphabetical order is displayed.

4 Click Prior. Person Information is again displayed.

Jump to Other Entries:

5 Click the element Name in the Attributes field. (It may be necessary to scroll the contents of the field to bring Name into view.) Click Jump. (Click Yes if you are asked if you want to save Person Information.) The repository entry for the data element Name is displayed.

6 Move to page two by clicking the Physical Information tab at the top of the dialog box. (The current page number is displayed in the upper right corner of the Define window.) This displays more information about the current entry, including the Location information that indicates where the current entry is used.

7 Click the line in the Location field containing Person Information. This highlights the line.

8 Click Jump. The entry for Person Information is once again displayed. The Locations tab (page 2) is currently displayed. An alternative to selecting Jump to switch to another repository entry is to double-click the entry name in the Location field or to click the Back button.

9 Move to page one by clicking the Description tab.

Search Capabilities

Searching for entries in the repository is an easy procedure. It can also be a very useful feature because you can set the Search Criteria to display only certain entry types as you move from one repository entry to the next. To search for an entry in the repository:

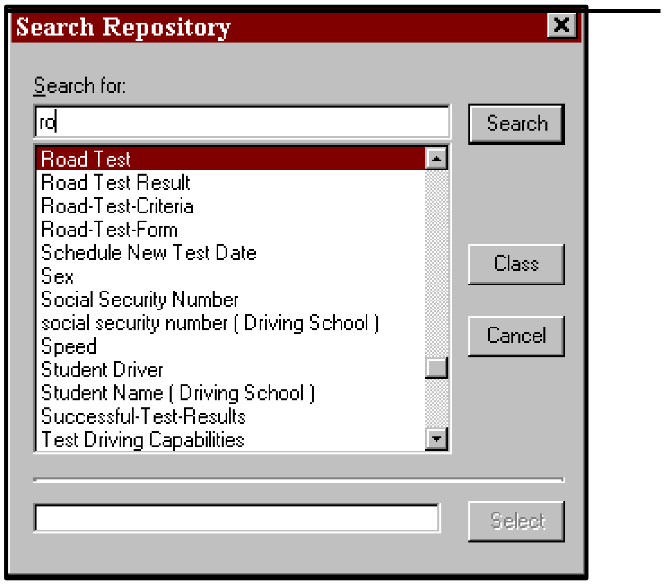
Access an Entry:

1 Click Clear. This clears the dialog box but does not delete the entry.

2 Type ―Road Test‖ and press ENTER twice. The repository entry for Road Test is displayed with all of the information that has been entered into the repository for this entry. (This was done for you in the samples included with the TEST project.)

3 Click Clear.

Search for the Entry: 4 Click the Search button to open the search box to select from the repository. Type ―r‖ and entries that begin with ―r‖ appear in the list box. If you now type an ―o,‖ you see that the repository searches incrementally as you type, getting closer to the entry you want.



Road Test
Road Test Result
Road-Test-Criteria
Road-Test-Form
Schedule New Test Date
Sex
Social Security Number
social security number (Driving School)
Speed
Student Driver
Student Name (Driving School)
Successful-Test-Results
Test Driving Capabilities

Figure 17-5 Repository Search Dialog Box

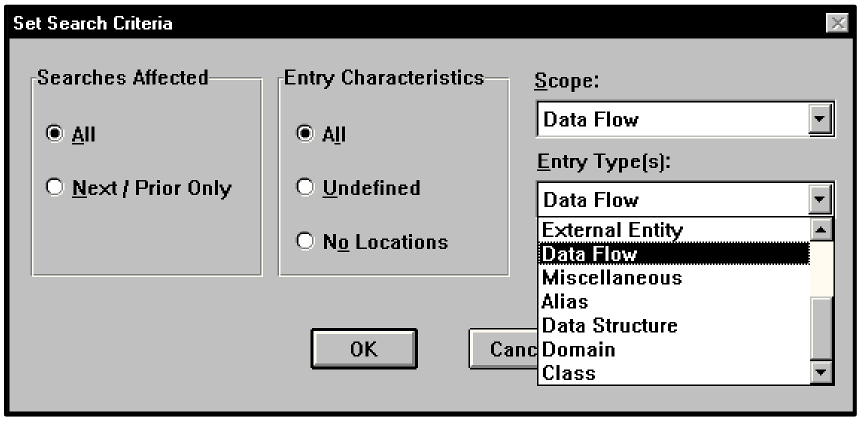
5 Click Road Test and then click Search. The repository entry for Road Test is displayed.

Setting the Search Criteria

Search criteria set the scope of the entries that are displayed as you search through the repository.

Clear the Dialog Box: 1 Click Clear to clear the dialog box.

Set the Criteria: 2 Click Search Criteria. You see a dialog box entitled Set Search Criteria, as shown in Figure 17-6.


Set Search Criteria

Searches Affected
- (●) All
- ( ) Next / Prior Only

Entry Characteristics
- (●) All
- ( ) Undefined
- ( ) No Locations

Scope: Data Flow

Entry Type(s): Data Flow
- External Entity
- Data Flow
- Miscellaneous
- Alias
- Data Structure
- Domain
- Class

Figure 17-6 Setting Repository Search Criteria

3 In the box entitled Searches Affected, select All. This is the method used to limit the scope of the entries displayed when navigating the repository using Next and Prior, as well as the entries that are displayed when you select Search.

4 In the box labeled Entry Characteristics, select All. This tells Visible Analyst to search all items in the repository, rather than only those entries that are Undefined or entries that have No Locations. No Location entries are typically those that have been entered directly into the repository rather than added to the repository by being placed on a diagram.

5 Click the down arrow on the right side of the field marked Scope. This allows you to choose the diagram type to which you wish to limit your search. Select Data Flow.

6 Click the down arrow on the right side of the field marked Entry Type(s). This allows you to be very specific about the type of entry to which you wish to limit your search. You can choose individual types and some combination types.

7 Select Data Flow, then click OK.

Try Out the Criteria: 8 At the blank Define dialog box, click Search. Because your search criteria limits searches to data flows, the list displays only the entries in the repository of the type data flow. Select Road-Test-Criteria and then click Search.

9 Now click Next. The next entry displayed is the next *data flow* in alphabetical order, rather than simply the next *entry* in alphabetical order. If you click Next a few more times, you notice that only data flow entries are displayed.

10 Click Search Criteria again and set Scope back to Entire Repository. Be sure that Entry Type(s) is set to All. Click OK.

Using Search to Add Items to a Field

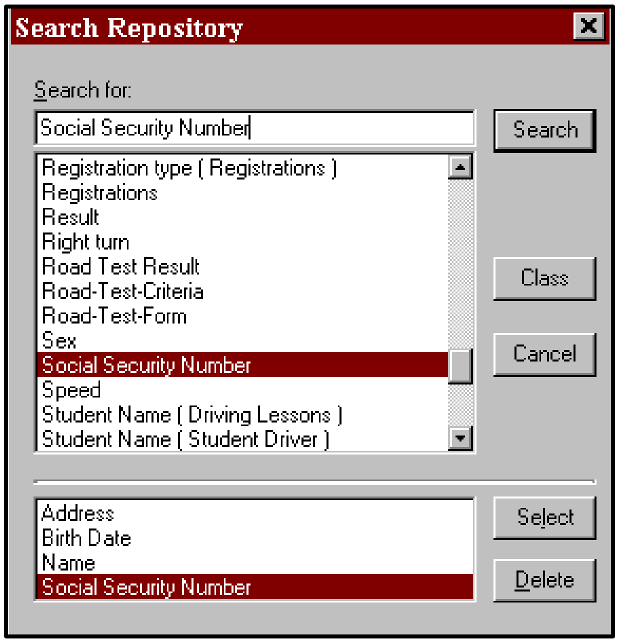
The Search feature can also be used to add repository entries to a field without retyping them. This option is very useful for adding multiple data elements to an Attributes field. Instead of typing the name into the field, you can select it using the Search function.

Clear the Dialog Box: 1 Click Clear.

Find an Entry: 2 Type ―V‖ in the Label field and click Search. Valid-Applicant should be the first entry on the list. Click on it and it appears in the Search For field. Click Search and the repository entry for Valid-Applicant appears.

Select Attributes: 3 Click on the Attributes field.

4 Click the Search button. The available data elements are displayed. Double-click on Address, Birth Date, Name, and Social Security Number. All the selected elements are displayed at the bottom of the Search dialog box as shown in Figure 17-7.



Social Security Number

Registration type (Registrations)
Registrations
Result
Right turn
Road Test Result
Road-Test-Criteria
Road-Test-Form
Sex
Social Security Number
Speed
Student Name (Driving Lessons)
Student Name (Student Driver)

Address
Birth Date
Name
Social Security Number

Figure 17-7 Add Information with Search

Add Attributes and Save:	5	Click Select. All the selected elements are added to the Attributes field. Click Save and then click Exit.

Where To Go from Here

OVERVIEW

This concludes the Visible Analyst tutorial. To exit the program, select Exit from the File menu.

You have now completed exercises in many of the major elements of planning, structured analysis and design, data modeling and object modeling:

- Drawing diagrams to model a system.
- Using methodology rules to insure against inconsistencies.
- Adding written definition to the graphic model.
- Embellishing the model following the initial layout.
- Expanding the model through definition of repository elements.
- Generating reports.

All structured software and systems engineering involve these same basic operations.

REAL WORLD APPLICATION

The example project that you created was a simple one. The real power of MDA is in the application to systems too complex to keep in your head at one time, too large to be reviewed by inspection, too widespread to have only one person working on the whole job. These types of projects include nearly every system designed today. That power shows itself in four areas:

- The assurance of accuracy and completeness, that no elements are left dangling or unaccounted for.
- The prompting and reminders that error checks and repository output provide, to focus attention in the midst of a dauntingly complex assignment.
- The word processing-like ease with which changes and modifications can be accomplished, while ripple effects are flagged and accounted for.
- The convenience of thorough documentation that is produced concurrently with the design, not as a drudgery-filled after-effort.

The power of MDA further multiplies into substantial gains in productivity, communication and quality when applied to team effort. The Corporate and Zachman Editions of Visible Analyst have significant capabilities for exporting and importing information, either through portable media or through participation in a local area network. This allows Visible Analyst to become an integral part of any development environment, sharing information and expanding the value of labor. Many groups thus can benefit from another group‘s hard work.

Finally, the application of the power of MDA productivity enhancement is not limited to software but can be applied toward analyzing and designing any system, such as:

Manufacturing	Medical Diagnostic Analysis
Planning	Command and Control Operations
Processing	Administrative Procedures
Legal/Judicial	Inventory Control
Audits	

Visible Analyst is designed to be a natural extension of the way you think, create, and analyze. Our goal is to ―Put MDA Within Everybody‘s Reach‖ and to make MDA tools and the methodologies of structured analysis, design, data modeling and object modeling a natural, seamless, integrated part of your everyday work, rather than an arcane ritual to be occasionally endured by specialists. The integration and flexibility to use the components and elements that you deem necessary for your application provide you with a customizable tool set that can be adapted to support how you choose to work on the design and development of information systems.

WHAT TO DO NEXT?

There are still many things for you to do to become comfortable and committed to the use of a MDA tool. We suggest the following:

- Study and review the logic concepts introduced in the tutorial (consider reviewing the referenced materials).
- Review the tutorial steps and practice any areas that were unclear.
- Select a personal real-world project to do using Visible Analyst. Make it a modest-sized effort to give you some time to explore and experiment with the tool.
- Select the parts and components that you want to use in your software development practice.
- Practice, practice, practice. Make adjustments where needed.
- Make the tool a regular part of all of your projects.
- Define standards and procedures for library components.
- Build disciplines and skills with concepts and Visible Analyst.
- Use technical support as needed - don‘t get stuck, don‘t become frustrated.

- Stop and evaluate what you have done, show others, review the work of others - find ways to improve the content and the processes.
- Practice reusability wherever possible.
- Stay up to date with the tool and the evolving methodologies.
- Build a library of materials for use in future projects.

CONCLUSION

We hope that these lessons have helped to make you feel comfortable with the planning, structured systems analysis and design, and data modeling tools, and their implementations in our product.

For more information or if you have any questions about MDA or structured analysis, please contact the Visible Systems technical support staff:

Telephone	(617) 902-0767
FAX	(508) 302-2400

Internet:
http://www.visiblesystemscorp.com
E-mail:
support@visiblesystemscorp.com

You can send your comments to:

Customer Comments
Visible Systems Corporation
24 School Street, 2nd floor
Boston, MA 02108