SECTION 7 – DOCUMENTATION GUIDE

Documentation should be written following specific standards defining the content, approach, and objectives. The standards provide a uniform plan for handling of documentation. ~~Binders~~ may be hardcopy or may be available as directories of document files available to authorized personnel over the LAN or (where documents are saved to HTML formats) over the Internet.

A. Feasibility Study Documentation

A feasibility study is conducted to determine the practicality of changing, developing or acquiring a system or computer program. Such studies may be omitted when a project is mandated by top management, the Legislature, and/or outside regulatory agencies.

Feasibility Study Binder Contents

A feasibility binder labeled with the title of the study will include the following:

- An accumulated table of contents.
- All memos related to the study.
- Vendor correspondence.
- Vendor literature.
- Accumulated costs data for the study.
- A log listing the calendar days and hours worked on the study.
- Documentation of all meetings.
- A copy of the final report.

The Final Feasibility Study Report

See the exhibit below for one format for reporting project proposal costs. Consistency of format, once established, is more essential than individual style and saves time when reviewing the contents.

Exhibit: Format for Project Proposal Costs

MAJOR PROJECT TASKS
(Use only those applicable to this project.)

	START DATE	FINISH DATE	ESTIMATED COST
Preliminary Analysis	xx/xx/xx	xx/xx/xx	$xxx,xxx
Systems Analysis/Design	xx/xx/xx	xx/xx/xx	$xxx,xxx
Programming	xx/xx/xx	xx/xx/xx	$xxx,xxx
Data and File Conversion Costs	xx/xx/xx	xx/xx/xx	$xxx,xxx
System and Program Testing			$xxx,xxx
Training Costs			$xxx,xxx
Hardware Purchase			$xxx,xxx
Software Purchase			$xxx,xxx
Administrative Costs			$xxx,xxx
Miscellaneous			
TOTAL COST			$xxx,xxx

ESTIMATED ANNUAL OPERATION COST

Data Entry	$xxx,xxx	
Processing	$xxx,xxx	
System/Program Maintenance	$xxx,xxx	
Administrative	$xxx,xxx	
Other (specify)	$xxx,xxx	
TOTAL ESTIMATED ANNUAL OPERATIONS COST		$xxx,xxx

B. Requirements Specification and Prototype Documentation

Requirements specification documentation is the recording of activities associated with systems analysis and design efforts. It provides proposed system design specifications in a format which management can easily evaluate on merit for approval. It also provides required specifications for software and vendor bidding. Requirements specifications documentation and its maintenance are vital to the success of a project and future systems maintenance.

Requirements Specification Documentation Binder

A requirements specifications documentation binder is maintained for each project but is not intended to replace the Work Plan documentation process. Forms in this section are examples of standard formats to be used as needed. This binder is maintained by the lead systems analyst for the project and contains:

A. Title page: The first or title page contains the project name and identification number, effective dates, and the lead analyst's name and title. Names and responsibilities of all other persons assigned to the project are listed on the second page.

B. Table of contents: This reflects the specific contents in the requirements specifications binder and requires updating as changes and/or additions are made.

C. Revision sheet: Every requirements specifications binder contains columns for:

- Number of revisions: Revisions are numbered sequentially starting with 1.
- Revision date: Effective date of the revision.
- Portion revised: The specific part of the system under revision, including all areas of documentation such as procedures, flowcharts, reports, etc.
- Name: The name of the systems analyst making the revision.
- Authorization: The name of the person authorizing the revision.

D. Systems Concept: The systems concept is prepared by the lead systems analyst from specifications provided by the requesting authority. This section includes copies of correspondence or written requests which outline the requirements of the system. Once approved, it serves as a guide for analysts to design the technical specifications of the system and includes:

- Approval: The first page lists the project name, the requesting authority's name, and the title of the person approving the project.
- Project history: This section states the background that justifies the project.
- Project objectives: The details of project objectives are presented. All functional areas affected by this project are listed, and any problems with crossed lines of authority are noted so that the required coordination can be spelled out. In addition, anticipated results from the successful completion of the project are detailed. To ensure complete understanding of the system by management and users, what will not be accomplished by the system is also noted.

E. Requirements: The documentation requirements necessary for a proper completion of the project are:

- A brief explanation of the work to be done during each phase.
- A schedule for inception and completion dates of each phase and the key tasks within each phase. To help picture the overall schedule, Gantt charting is recommended for key tasks and phases of complex projects.
- Special requirements for project software from vendors and/or for in-house programmed software.

- Special requirements for outside acquisitions such as equipment, consultants, contract personnel and/or vendor services.
- Current in-house equipment resources that will be committed to the systems project.
- Man-hour resources, by names and titles, to be committed to the project.
- Resources expected from the requesting authority including hardware and personnel assistance such as training of extra manpower during conversion of the system.

F. Systems Specifications: Systems specifications provide a means of communication between the designer and the software providers who can be in-house programming staff or vendors. They provide a record of the structure, functions, flow, documentation and control of the system and include:

- System narrative
- Current date.
- Descriptive title of the system.
- General requirements with a description of each and with sources cited.
- Source, expected schedule, and volume of input data.
- Size and characteristics of the data.
- Sample exhibits of input documents.
- Outputs indicating volume and due dates. This may be a monitor display and/or hard copy.
- Sample exhibits of outputs.
- References to material or organizations that may furnish additional information. This may include vendor-supplied information.

G. System flowcharts: Flowcharts identify the processes performed and their sequences. Each system flowchart is contained on one page; if greater detail is required of any given area of the system, a subsystem flowchart is drawn which both non-technical management and computer programmers are able to understand.

H. Program specification requirements: These define the environment within a system in which a subsystem or a single program operates. They describe:

- Inputs and outputs, including source, format, disposition, retention method, and time period.
- Process, controls, functions, and limitations. These specifications prepared by the systems analyst to be used only by the programming staff, need to be confirmed by the person responsible for programming that they can be accomplished by this staff.

I. File/Database specifications: A complete and detailed specification of the files and/or database used in the system is identified. A reference sheet describes the file layout, field definitions and source references.
The file layout contains:

- File name.
- File number or label.
- Current date.
- File label.
- Record size.
- Storage medium.
- Record length required.
- Blocking factor (tape only).
- Retention requirements (daily, monthly, etc., and the expected currency is the number of days between the cutoff date and the completion of the update). "Monthly 5 days" means the file is updated monthly with 5 days required to complete the update.

- Security level.
- File sequence.
- Remarks.

J. Input description

The format for all the data going into the system. It provides the following information for each input record:

- Name.
- Date.
- Each field identified within a record, its type and length.
- Number of records per cycle (day, week, etc.)
- Disposition after it is received by the system.

K. Output description

The layout of each output should be displayed in monitor or hard copy format and is part of the systems documentation. Forms designed for this purpose include:

- Name of report.
- Number of report.
- Location and length of each item within a report layout.
- Number of copies.
- Relationship of items in the report to the source of the item information in the input description.

C. Program Development Documentation

Software (program) documentation serves various functions. First, program maintenance has become a major expense. The investment and effort required to document software is rewarded by reduced maintenance costs. Next, it communicates to the systems analyst or project leader that the programming has been properly done. Program documentation also provides information systems operations and users the information needed to operate the software. It also furnishes the auditors with information about the controls that have been established. Finally, it shows systems management what exactly has been accomplished with the money spent.

Software Binders

Information about developed programs and purchased software is maintained in "software binders," one for each written program or purchased software package. They contain:

A. General information

This section identifies the software and its objectives. Each program should have a unique identification number. The programs may be grouped together using any of the following categories:

- Low-level written programs.
- High-level written programs.
- Reusable code.
- CASE tools.
- Operating software.
- Database software.
- Network software.
- Purchased mainframe programs.
- Purchased workgroup software.
- Purchased microcomputer software.

B. General objectives: The general objectives include a brief narrative statement of each program's purpose or use.

C. Purchased software: Any license agreements connected with the programs, including the number of copies purchased and the last date of purchase. Any "Beta" operating software will be so identified, and its agreement information placed in this section.

D. Program's systems specifications: Vendor-supplied information will also be included.

E. Program flowcharts: Flowcharts will be flowcharted using available software that employs the ANSI (American National Standards Institute) symbols.

F. Written program documentation

- Source code listings.
- Source of input and format.
- Pre-alpha naming conventions. These are needed for the database, inputs, outputs and controls.
- Hard copy layouts.
- Screen display layouts.
- Screen layouts displaying control buttons, text boxes, and screen labels.
- Database layouts, definitions of field items, and naming

references.
- Table layouts.
- Data element cross-references.
- Subroutines, sources, and listings.
- Table lookup tables.
- Index of table field names (the names that can be found in database, programs, and input/output locations). Useful when programming and doing maintenance later.

G. Operations Specifications

The programmer or programmer analyst is responsible for preparing operating instructions that can be used by the central computer operations, staff, client/server users, work group systems, end user systems, or standalone users. This documentation should contain detailed instructions on how to operate the program and handle exceptions that may occur, and will require the signed and dated approval of the highest authority for the unit for which they are intended. LAN, WAN or client/server user instructions will require the approval of the head of information systems or his/her designated representative. The types of information may include:

Cover page information
- Program number.
- Program description or name.
- Name of programmer or help desk contact.
- Type of program.
- Priority requirements.
- Estimated processing time.
- Date prepared and revision dates.

Input/Output requirements
- Keyboard instructions.
- Scanning instrument instructions.
- Removable disk and/or CD instructions.
- External label.
- Internal label.
- Printer(s) required.
- Form number.
- Report disposition.
- Other input/output devices not listed.

Magnetic tape instructions:
- External label.
- Internal label.
- Input or output usage.
- File protection requirements.
- Tape disposition.

General operating instructions.

Restart procedures.

D. Testing Documentation

Program Test Data

Listings of input used for testing and the resulting outputs are kept in the program test file. Monitor displays are screen printed or drawn for documentation.

E. Implementation Documentation

The operating procedures necessary to initiate, schedule, process, control, and restart all phases of the system(s) while it is in operation are maintained in a binder for the operations personnel. All the documentation must be kept current. The project manager is responsible for compiling and validating new systems. Maintenance information systems personnel must update existing manuals. The general operations to keep in mind that require policy and related documentation are:

- Backup and recovery procedures
- Computer operation procedures
- Terminal user procedures
- Microcomputer operation procedures

Flowcharting Standards

Flowcharts perform several functions in the systems investigation, analysis, design, development and installation phases. Flowchart diagrams range in complexity from global system flowcharts to procedure step detail flowcharts. Consider the level and helpfulness of these different representations in deciding which of these, if any, will be useful in your analysis and presentation. The levels of flowcharting are:

A. Workflow Diagram

In the analysis stage, the work flow diagram is used to document and analyze the flow of work within critical business functions only-those that are essential to meeting the organization's business objectives. In later stages, the work flow diagram is used to support the design of new processes, jobs and tasks, and should therefore be more complete. The contents of a workflow diagram include:

1) Start-The starting point of the process. If applicable, describe the event or condition that initiates or triggers the process (end of day, customer inquiry, etc.).
2) Document-Hard-copy document used as input to or output from a component.
3) On-Page Connector-A connection of flowchart components with others on the same page.
4) Component-An action step in the process.
5) Decision Point-Different branches in the work flow based on a specific condition.
6) Arrow-The process flow from one component to another.

B. Dataflow Diagram

Dataflow (data flow) diagrams are replacing the more traditional systems flowchart. Like its predecessor, it has not been guided by any one national standard. There is more than one proposed set of symbols, but all share the same concept and objective. Data flow diagrams, have many advantages over traditional systems flowcharts when presenting graduated levels of overviews for both current and proposed systems. Its simplicity provides for a more uniform pictorial presentation of systems that can be followed by systems analysts, programmers, operations personnel, and most users. There are commercially available dataflow diagram templates and microcomputer drawing software.

C. Event Modeling

An event is an external occurrence that is of interest to the application system. Events represent one aspect of the requirements model; combined with data and processes, they constitute a comprehensive and verifiable definition of application requirements. Although event modeling represents a relatively new form of modeling for business systems, events have been used for many years in real time systems.

Event modeling is an "outside in" approach to analysis. It focuses on understanding the external

environment: business, procedural and otherwise. Events express the system's perception of external activities; therefore, event modeling allows the external environment to influence the requirements definition, rather then basing the environment on the application's structure. This approach supports the application's consistency with the workflow and other business activities.

Events are external to the system; they represent the requirements the system must recognize and the responses the system must produce. An event is not an externalization of an internal process. In other words, "Enter Customer Information" is not itself an event. Rather, the event is the action that causes the system's need for customer information.

Events support the systems building process in all stages of the systems development lifecycle. This support revolves around the use of events in the definition and clarification of user requirements. Events define user requirements in concise business terms, and are therefore an excellent means of communicating requirements to the users and other members of the project team. The comprehensive nature of the event model (it includes all external actions the application system will support) defines the scope of the application system. Finally, the business nature of the event model and its definition of business activities aids testing.

D. Context Diagram

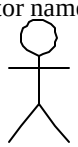
There are a number of approaches to defining the scope of an application system; one of the most effective is a context diagram. A context diagram is the highest level definition of an application process. It lists the application as a single object surrounded by all its external entities. Information flowing from the entities to the application (for example, "places ads") represents an event. Information flowing in the opposite direction (for example, "issues paper") represents the application's response. The context diagram identifies the overall scope of functionality the business will support. The analyst can use event modeling to create the context diagram.

F. Object Oriented Deliverables

The following examples were developed using Rational Software's Rational Rose modeling tool. The diagramming conventions outlined are those defined by the Universal Modeling Language (UML) and supported by Rational Rose. If other object oriented modeling tools are used the diagramming conventions might differ slightly.

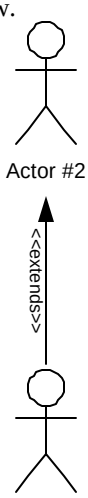
A. Actors List

An actor defines a coherent set of roles that external entities, such as humans, machines, or other systems, can play when interacting with the system being developed. An actor is depicted as a stick figure with the actor name below the figure, as shown below.



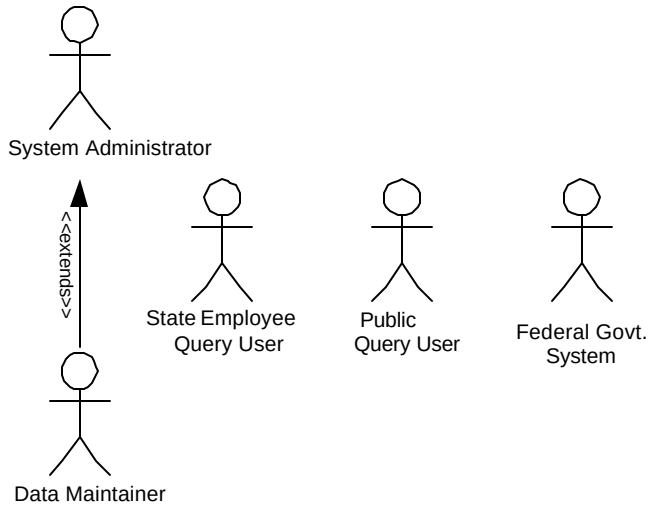
Actor Name

An actor case can be related to another actor through an *extends* relationship. An extends relationship between actors may be used when one actor (Actor #1) may be able to perform a set of use cases, and another actor (Actor #2) may perform the same set as the first, but also may perform one or more additional use cases that the first cannot. A generic example of an extends relationship between actors is shown below.



In the example provided above, Actor #2 can do everything that Actor #1 can, but Actor #2 may also do additional things that Actor #1 cannot. Actor #2, therefore, extends Actor #1.

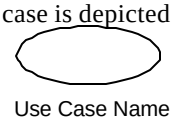
An actors list identifies all actors pertaining to the system being modeled. For example:



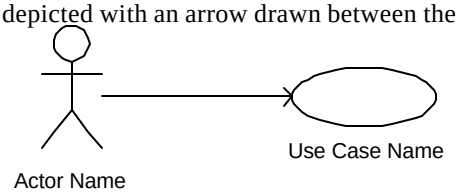
B. Use Case Model

Use case models seek to represent the functionality of a system from the perspective of entities interacting with the system. They provide an external view that is independent of the internal structure of the system. A use case model is composed of one or more use case diagrams, the key elements of which are *actors* and *use cases*.

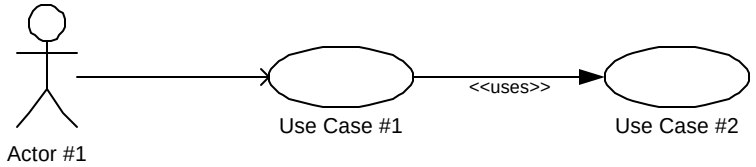
A use case is a sequence of steps or transactions performed by an actor in dialog with the system. A use case is depicted as an oval with the use case name below the oval, as shown below.



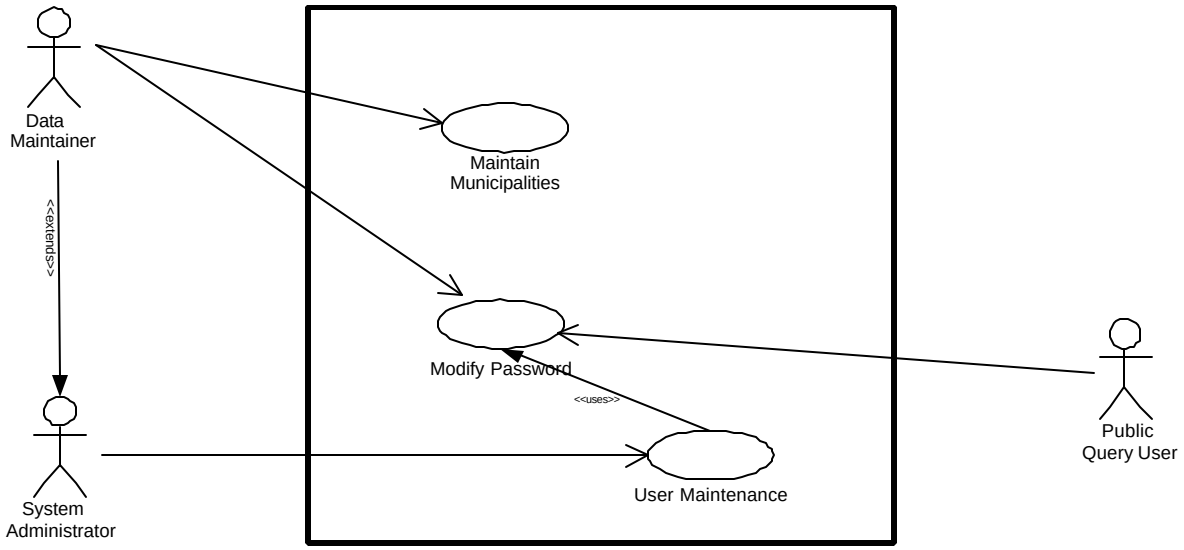
An actor communicates with a use case to utilize the system in a specific way. The communication is depicted with an arrow drawn between the actor and the use case, as shown below.



A use case can be related to another use case through a *uses* relationship. A uses relationship allows common behavior to be packaged into separate use cases that may be shared by other use cases. For example, one use case may use another use case as part of its process, however the second use case may also be used independently of the first by an actor. A generic example of a uses relationship between use cases is shown below.

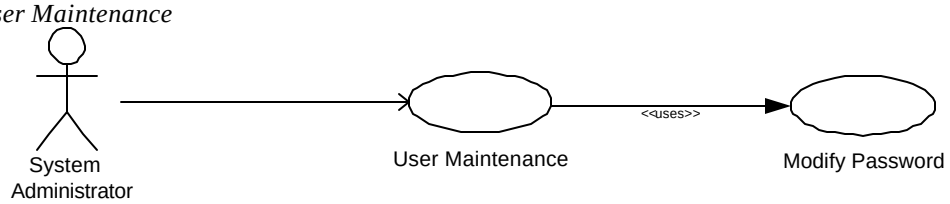


A use case model depicts the functionality for the system being modeled. For example:



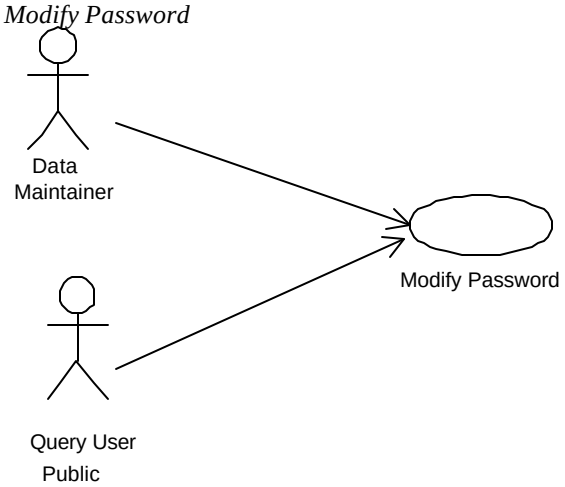
Subsequent sections break this model down into its component use case diagrams. A brief explanation accompanies each use case diagram.

User Maintenance



The System Administrator may add/maintain users through the User Maintenance process. Users can be added or removed, roles can be assigned to or taken away from users based upon business needs, and passwords can be set or reset using the Modify Password process.

Modify Password



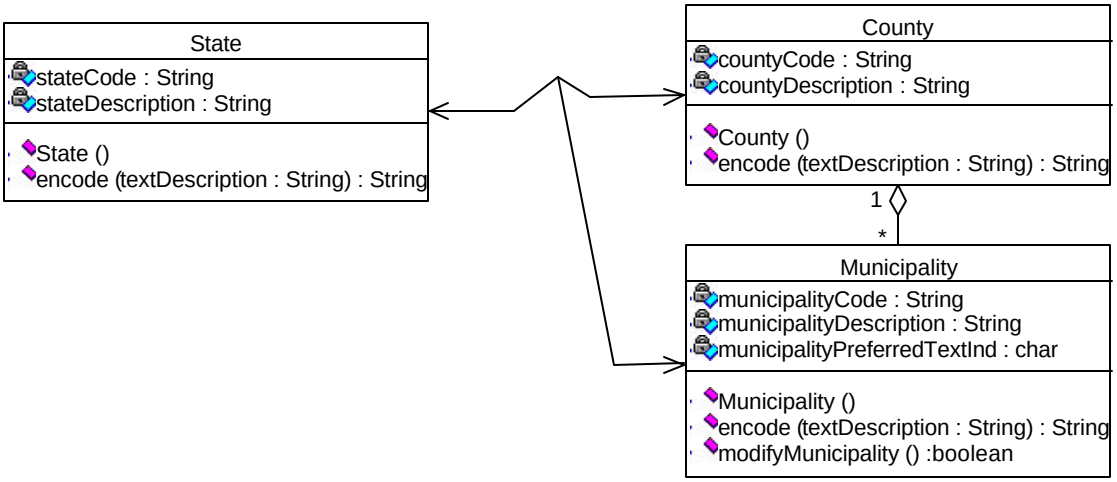
The Data Maintainer and the Public Query User may change their system login password through the Modify Password process.

C. Class Model

A class model, also known as a business object model, represents a structural view of the real-world business concepts for the system under development. A class represents a real world concept within the problem domain being analyzed. A class may have data, behavior, and relationships to other classes.
To aid in interpreting the class model, diagramming conventions are described here.

Convention	Description
County countyCode : String countyDescription : String County( ) encode( )	Each class is represented by a box divided into three sections. The top section contains the class name. The second section identifies the data, or *attributes,* of the class. The third section identifies the class behavior, or *methods*. A class can be thought of as a template from which *objects* are created, or *instantiated*. An *instance* of the County class is an object that contains information about a single county. The methods on that object can interact with the object's attributes.
countyCode : String	This convention is used to identify the *type* of the attribute. The type can be thought of as being similar to datatype, although object orientation allows any class to be a type.
	The lock shown beside an attribute identifies the attribute as *private* to the class. The value of the attribute is only known to methods of the object itself. Methods of other objects cannot interact directly with a private attribute.
	The key shown beside an attribute identifies the attribute as *protected* to the class. The value of the attribute is only known to methods of the object itself or by methods of subclass objects.
encode(textDescription : String) : String	The convention used to identify methods. Methods can be thought of as being similar to procedures or functions. Information may be included within the parentheses identifying parameters that are passed to the method. The return type may be specified outside the parentheses. In the example provided, a string text description is passed as a parameter to the encode method, and a string is returned.
	Indicates the method is public and can be invoked by any other object.
	Indicates the method is protected and can be invoked only by methods of its own object or by methods of subclass objects.
	A line ending with an arrowhead represents a relationship between two classes. This *association* means one class uses another class.
	Identifies a generalization relationship. The point of the triangle is attached to the more general, or *base class*. The line connects to the more specific, or *subclass*. The relationship is read "the <subclass> *is a* <base class>." For example, a mile marker is a feature.
	Identifies composition relationships, in which instances of the component class must all belong to the same container object. The diamond is attached to the container class and the line connects to the component class. The relationship is read "the <container class> *has a* <component class>." For example, a county route has a feature.
1 *	A number or an asterisk at the end of the relationship line represents multiplicity. It represents the number of instances of a class involved in the relationship. The asterisk represents an infinite number. In the example, one instance of the class on the left end of the relationship can have infinite instances of the class on the right end of the relationship. For example, one county route can have infinite features.

A class model depicts the classes and relationships that will be used to represent the system being modeled. For example:



Municipality

The Municipality class contains information pertaining to municipality names that can be maintained by the Data Maintainer. The Municipality class is a component of the County class, and is used by the Encoder class to convert text descriptions of municipalities into code values.

Attributes

Name	Type	Description
MunicipalityCode	String	Municipality code
MunicipalityDescription	String	Municipality description
MunicipalityPreferredTextInd	Char	Municipality preferred text indicator

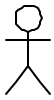
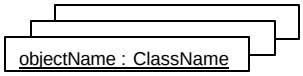
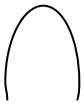
Operations

Name	Description
Municipality()	The constructor method for the class
Encode()	When given a text description for a municipality, provides the corresponding code value
ModifyMunicipality()	Provides a means to maintain municipality data

D. Interaction Model

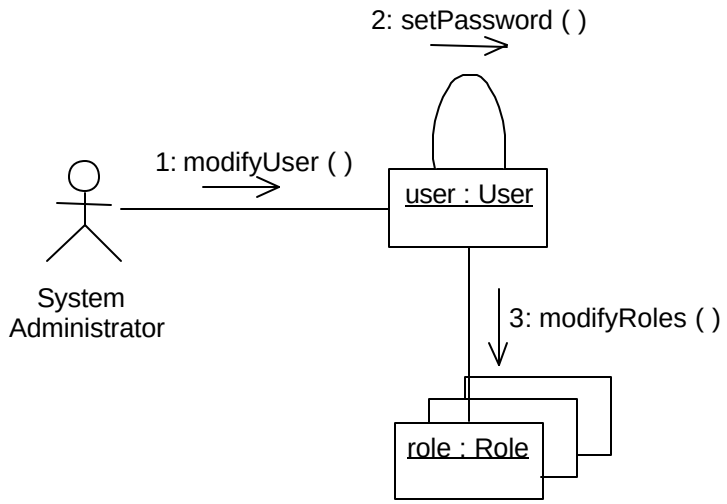
An interaction model represents how objects or instances of classes defined in the class model interact in order to carry out specific scenarios described by the use case model. As such, it provides the link between the requirements (use case model) and the evolving class specifications (class model). The interaction model can be depicted as a collaboration diagram or a sequence diagram. A collaboration diagram is an interaction model that shows the sequence of messages that implement an operation or a transaction, whereas a sequence diagram traces the execution of an interaction in time. Since the sequence of events tends to be more important than an event timeline for most business applications this document focuses on collaboration diagrams. The diagramming conventions are described here to aid in interpreting interaction-collaboration models.

Convention	Description

 Actor Name	As in the use cases, an actor defines a coherent set of roles that external entities, such as humans, machines, or other systems, can play when interacting with the system being developed. An actor is depicted as a stick figure with the actor name below the figure.
objectName : ClassName	This convention is used to identify an object (objectName) and its class (ClassName) separated by colon (:). An object has state, behavior, and identity. The structure and behavior of similar objects are defined in their common class. An object is an instance of a class.
 objectName : ClassName	The multiple objects that are instances of the same class are represented using this convention. It is used when multiple instances of a class are necessary to perform an action.
	This indicates a link between two objects. Objects interact through their links to other objects. A link is an instance of an association, analogous to an object being an instance of a class. The existence of a link between two classes symbolizes a path of communication between instances of the classes: one object may send messages to another. Links can support multiple messages in either direction.
. 	This indicates a self-link by an object.
1: (checkVal) messageName (param)	This convention indicates a message between two objects, or a self-message. A message conveys the source object's invocation of an operation of the destination object. Messages are carried by links. Messages can be of forward or reverse type, depending on which object is invoking the operation. Messages are preceded by a number indicating the operating sequence (1, 2, 3, etc.), then a possible constraint to be checked (checkVal) prior to performing the action. Messages can have multiple parameters separated by commas, passed in parentheses after the message name.

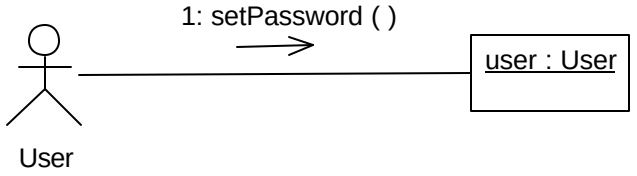
The following section headings should correspond with the use cases provided earlier. Each section includes a collaboration diagram as well as a sequential narrative description for each diagram. The diagrams and descriptions are not intended to be fully comprehensive nor do they address all possible scenarios for a given process. The information provided is an overview of each process and the main classes and methods it will use.

User Maintenance



1. The System Administrator may add, delete, and/or modify users in the system by using the modifyUser() method of the User class.
2. The setPassword() method of the User class may be invoked to create a password for a new user or change the password for an existing user.
3. The modifyRoles() method of the Role class may be invoked to assign roles to new users or change roles of existing users.

Modify Password



1. Any user can reset their password by using the setPassword() method of the User class.

E. Physical System Architecture

The primary goal of the Physical System Architecture section is to identify the processes (i.e., classes) that are required to support the business requirements within the scope of the automation boundary of the system under development.

Physical System Architecture - Logical Process to Physical Class Mapping

This section maps the logical processes identified in the business requirements definition document to the physical classes described earlier in this document. The classes provide a mapping to the physical implementation of the program logic. The classes specify the attributes and the methods that their instance objects can know and perform.
These logical processes can be mapped to the physical classes identified previously in this document as depicted in the following table:

Mapping of Logical Processes to Physical Classes

Logical Process	Physical Class	Description	Implementation Language
User Management	User Role UserManagerUI	This group lists the classes used to manage users, passwords, and roles.	Java and embedded SQL

Physical System Architecture - Data Access

Data access classes should be created to facilitate interaction between persistent storage and the physical classes discussed in this document. All data access will be performed through the data access classes in accordance with the established technical architecture, with one exception—data access for reporting purposes. Reporting needs are most easily met via a dedicated reporting tool. Most of the reporting tools on the market currently access the data in persistent storage directly.

Physical System Architecture - Common Services

A list of common services identified should be provided as part of this document.

Physical System Architecture - Interfaces to Other Systems

All system interfaces should be identified and described as part of this document. Communications formats, schedule of communications, and error determination and recovery need to be addressed.

Physical System Architecture - Auditing and Archiving

Any archiving and/or auditing requirements should be detailed as part of this document.

Physical System Architecture - Object/Database Interaction Matrix

The CRUD matrix shows the interaction between the classes and the database tables. The definition of the attributes in the table are listed:

- C = Create The process may add a record to the data store.
- R = Read The process may retrieve data from the data store.
- U = Update The process may modify data in an existing record.
- D = Delete The process may permanently delete a record from the data store.

The following table depicts the CRUD matrix. The GUI classes are not listed in the table, since they do not interact directly with the database tables, nor are the data access or report writer-related classes for reasons stated prior. This CRUD matrix is intended only to be representative of which classes can perform what processes to which database tables. In reality, although not depicted here, only the data access classes and the report-writing classes will interact with database tables.

Matrix of Database Actions Allowed by Each Class

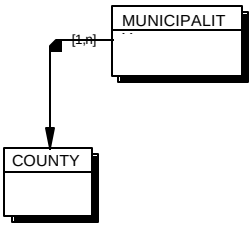
Classes\Tables	**STATE**	**COUNTY**	**MUNICIPALITY**
State	CRUD		
County		CRUD	
Municipality			CRUD

E. Physical Data Model

The physical data model identifies the data structures required to support the application. It identifies the physical data tables and column names derived from the logical data model, the data structures required to support interfaces with existing systems, and any additional tables required to support the application, including user access and error logs. This section also identifies the table indexes the application will use. The conventions used in the physical data model are somewhat different than those used in the logical model, and are described here to aid in interpreting the diagram.

Convention	Description
→	Arrows represent relationships between tables, and point toward the parent table.
[1,n]	Cardinality describes the number of child records a parent record can have. In this model it is represented by values in square brackets, located on the relationship arrow. Cardinality is read as "the parent has 1 to *n* child records." In this example the parent must have at least one child record.

Entity Relationship Diagram



Each of the relationships identified in the model is described below.
County and Municipality

- Each County must have one or more Municipality record.
- Each Municipality record is identified to one and only one County.

Table Descriptions

The following sections describe each physical table including column definitions, business rules, and table indexes. Conventions are identified here to assist in interpreting the descriptions.
Table descriptions are presented in the following format:

Column Name	Logical Name	Datatype	PK	FK	Req'd
CNTY_CD	County Code	CHAR(2)	X	X	X

The column headed *PK* identifies columns that are part of the primary key. *FK* identifies columns that have a foreign key reference (a relationship to another table). *Req'd* identifies columns which must contain a value (cannot be NULL).
Following the table description are the column definitions listed in the order they appear in the table. Where appropriate, business rules are included with the column definitions. The business rules identified are used to assure data integrity for initial conversion and load, and for user maintenance of data. Business rules are not enforced by the database. They are enforced through application logic or operational control. They are presented here for review and approval.
Table indexes are identified in the following format:

Index Code	P	F	U	Column Code	Sort

ROAD NAME

COUNTY_PK_IDX	X		X	CNTY_CD	ASC

The column headed *P* identifies indexes created from Primary Key columns. *F* identifies indexes created from Foreign Key columns. *U* identifies unique indexes.

COUNTY Table
The County table contains the master list of North Carolina counties, and is used primarily as a validation table.

Column Name	Logical Name	Datatype	PK	FK	Req'd
CNTY_CD	County Code	CHAR(2)	X		X
CNTY_NM	County Name	VARCHAR2(30)			X

CNTY_CD
The code value that represents a county in North Carolina.

CNTY_NM
The text name of a North Carolina county.

G. Distributed System Architecture

The objective of this section is to distribute the individual data tables, defined in the *Physical Data Model* section, and classes, identified in the *Class Model* and *Physical System Architecture* sections, to appropriate client and server nodes of the technical environment.
An application might distribute its components based upon categories: GUI components, business rule components, and data access components. Each component will correspond to a physical class. All the GUI components could reside on client workstations; business rule and data access components could reside on the server. Database tables could also reside on the server.

Class Distribution

A mapping table for all the classes identified in the *Class Model* and *Physical System Architecture* sections is provided below. The following abbreviations are used in the table that follows.

BR - Business Rule
GUI - Graphical User Interface
C - Client
S - Server

Distribution of Physical System Architecture		
Class	**Component Type**	**Client or Server**
User	BR	S
UserManagerUI	GUI	C

Table Distribution

Either provide a statement here that all the database tables reside on the server or provide a table similar to above that details which database server each table will reside in. Distributed Data Architecture

H. Shared Services Meta-Data

The following Shared Services documentation is based on the documentation format and content planned by
the IRM. A statewide component repository is being developed where information about shared services can be stored and searched. While this repository is not available, this documentation should be kept in a DENR database for internal sharing.

1. Shared Service Name - The name assigned to the shared service in the NC Service Broker LDAP:

   - Business Service
   - Business Component (optional, information only)
   - Business Element (optional, information only)

2. Summary description of the Shared Service (provide business background and objectives)

3. Procedure for requesting and granting access to this service

4. Cost of subscribing to the service

5. Contact list

   5.1. Shared Service Owner
   5.2. Person authorized to grant access to the service
   5.3. Person authorized to process service change requests

6. Availability and Performance Levels

7. Security Requirements

8. Technical Environment

   Eg.

   Platform: ITS MVS Mainframe
   LPAR: "E"
   CICS Region: ZDOTPRD
   Language: COBOL, CICS
   Trans-ID or Executable: X200

9. Request Message Description

   9.1. Message Layout and List of data elements in the request message
   9.2 Descriptive data element name (do not use program reference names, eg. copybook field names)
   9.3. English language description of the data element
   9.4. Length of the data element in bytes (character format)
   9.5. Data element format (usually character)
   9.6. Mandatory or optional
   9.7. Valid data element values

   Example:

   Data element: Customer number

Description: This is the number assigned as the primary reference number associated with a North Carolina driver or citizen. This may either be a driver's license number, NC ID number, or a merged customer number (???). This is the primary key used in the Driver's License System database.
Length: 12 bytes
Format: character only
Presence: Optional. See full service description for combination of search parameters required.
Valid values:

- *characters a-z, 0-9*
- *must be right justified*
- *leading spaces must be initialized to be BLANK*

10. Response Message Description

    10.1. Message Layout and List of data elements in the response message
    10.2 Descriptive data element name (do not use program reference names, eg. copybook field names)
    10.3. English language description of the data element
    10.4. Length of the data element in bytes (character format)
    10.5. Data element format (usually character)
    10.6. Data element-level restrictions if any

11. Full description of the Shared Service or component, including business rules and processing logic.

12. Service Return Codes

    12.1. Code
    12.2. Text
    12.3. Recommended action

13. Other considerations

14. Clients currently using this shared service

    14.1. Agency/Division
    14.2. Application
    14.3. LDAP Client Name
    14.4. Agency Contact
    14.5. Daily Usage volume (no. of requests, no. of responses)

15. Service Testing Pack available to other agencies and divisions

    - Test drivers
    - Test data
    - Test cases and procedures
    - Where located

15. Effort hours to develop this service

    - Analysis
    - Design
    - Coding and Testing
    - QA and Documentation

16. Keywords that may be used as search criteria

End of Section 7