

OneTouch Solutions Developer's Guide

Emin Tolga Akgöz	1297373
Swaleh Kavuma	1272079
Doguhan Ilgaz Dikmen	1295070
Volkan Onur Gürsoy	1128891

1.Introduction

Our system has two main software parts: **Door Controller Software** and **Door Security Manager**. In this guide we mainly concentrate on the features of **Door Security Manager** software. **Door Security Manager** is a software program that configures and manages a door access control system using door controllers with a wireless network connection. This program is designed to make the access control system configuration and operation as simple as possible. This is done by breaking the configuration and operation processes into logical sections. Each section is responsible for a specific operation.

Door Security Manager has two distinct audiences:

Administrators — individuals who defines system parameters, manages users/doors/permission groups and transaction files, sets states of the doors and prepares reports

Developers — individuals who wish to extend **Door Security Manager** to create highly customized solutions. This audience is likely interested in creating highly reusable custom code that makes **Door Security Manager** do something new and interesting.

This guide is intended to document **Door Security Manager** for the second audience, developers, as defined above. If you fit more into the "administrator" audience defined above, you'll probably want to start by reading the **Door Security Manager** User Manual.

This document describes **Door Security Manager**'s services to the developer from a mostly visual point of view using UML diagrams. This guide is not a complete reference to the **Door Security Manager** software but rather a practical guide to applying **Door Security Manager**'s services to develop and deploy your own web applications. Beside these, we will provide the **Door Controller Software** features.

This guide covers the following topics:

Software Architecture- gives an overview of the system
Class Diagrams- introduces fundamental classes and their relationships
Database Model- model of the permanently stored objects in our system
Business Classes- classes related to application logic
Door Controller Software- the pseudocode of the PIC software

2.Software Architecture

As we said before, the system actually consists of two different software:

- **Door Security Manager(Master Computer Software):** This is the software which allows the customer to manage the system.It will run on a desktop computer running Windows Xp or Linux.
- **Door Controller Software:** The software that controls the user access through the doors.It will run on a embedded computer running Linux as an operating system.The distribution of the Linux would be prepared by us according to Door-Controller Software's requirements.

Considering these elements, the packages of the system and relationships among them are depicted below:

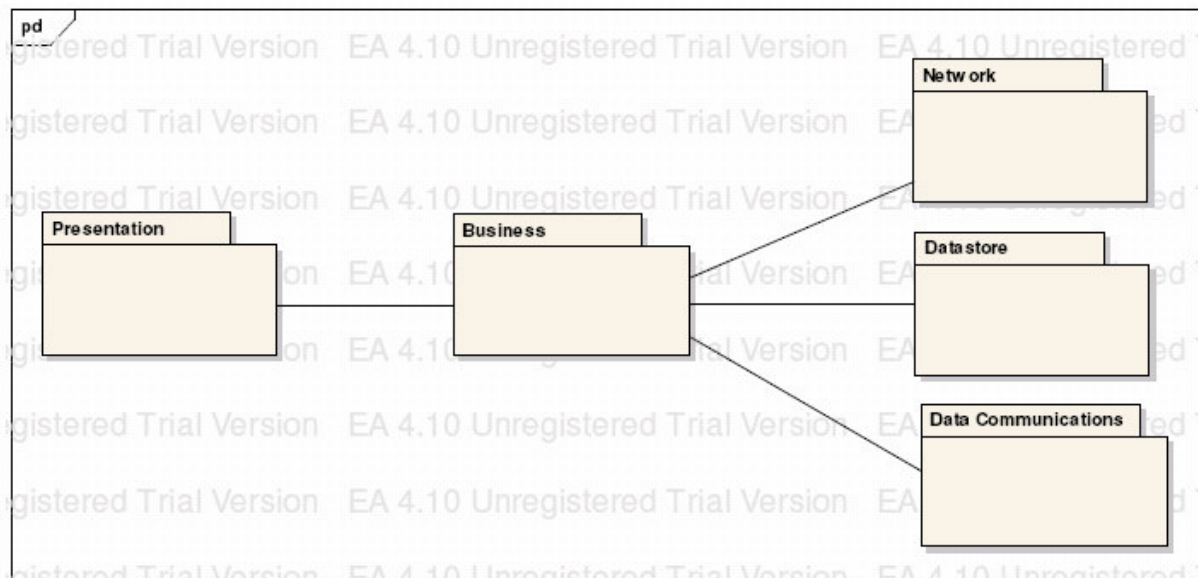


Figure – 1 : Packages

Presentation Package: Contains the GUI elements of the system, mainly door security manager's interface.

Business Package:Classes related application logic fall under this class.

Network Package:Classes related to network communication fall under this class.

Datastore Package:Classes related file and database management included in this class

Data Communications Package: Classes related to interfacing with physical devices belong to this class

3. Class Diagrams

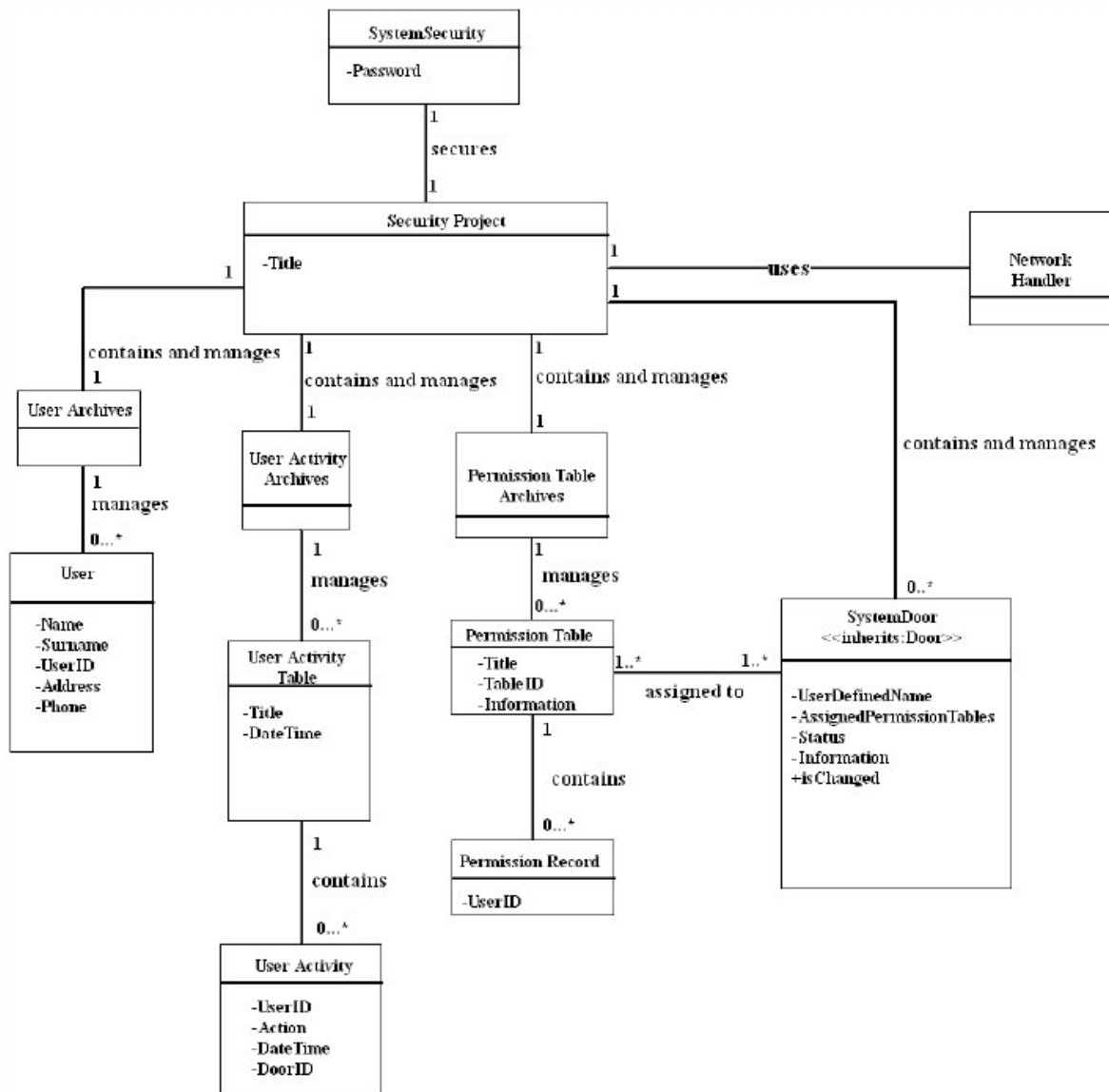


Figure-3:Class Diagram for the Door-Security Manager

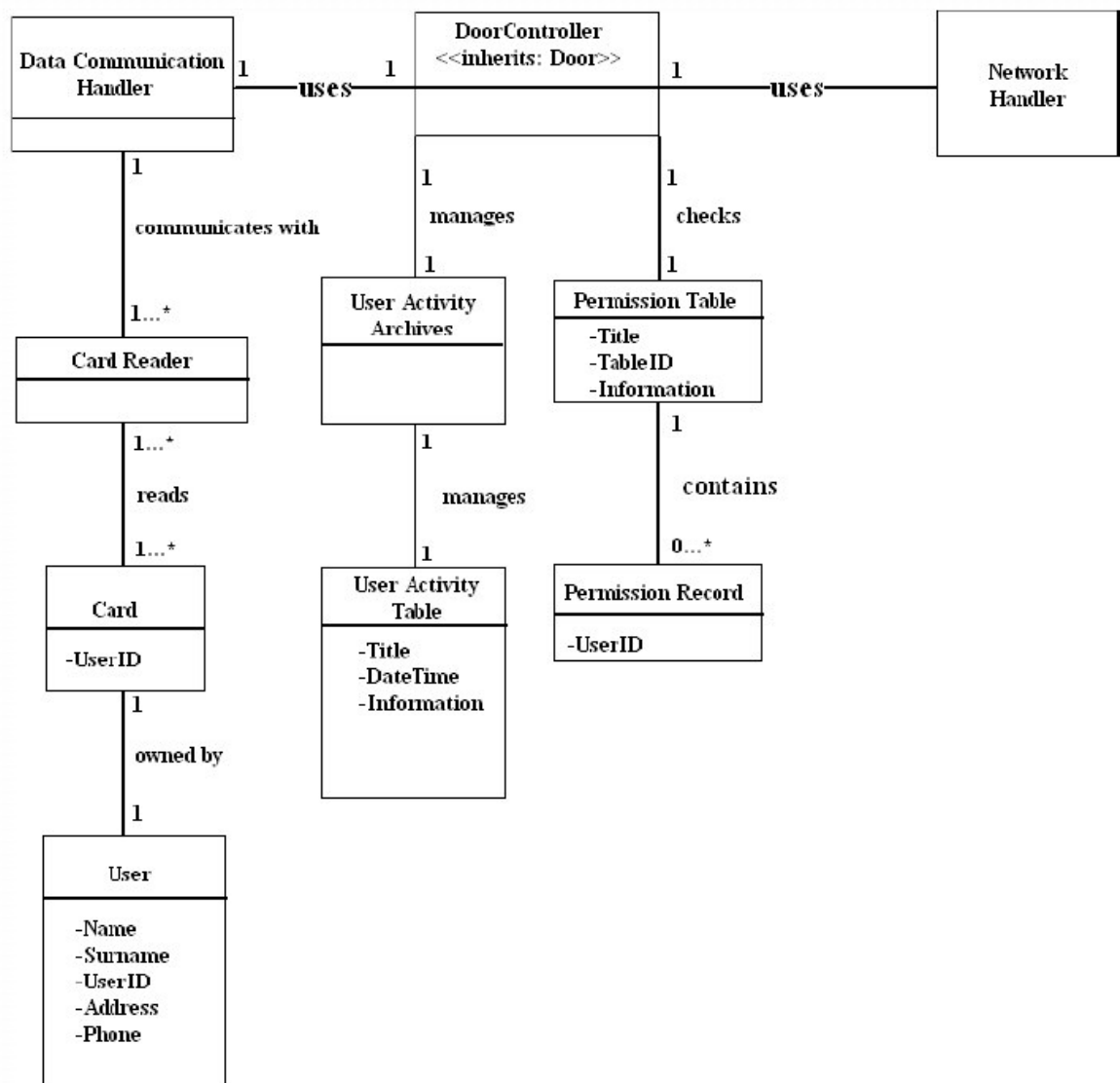


Figure-4:Class Diagram for the Door-Controller Software

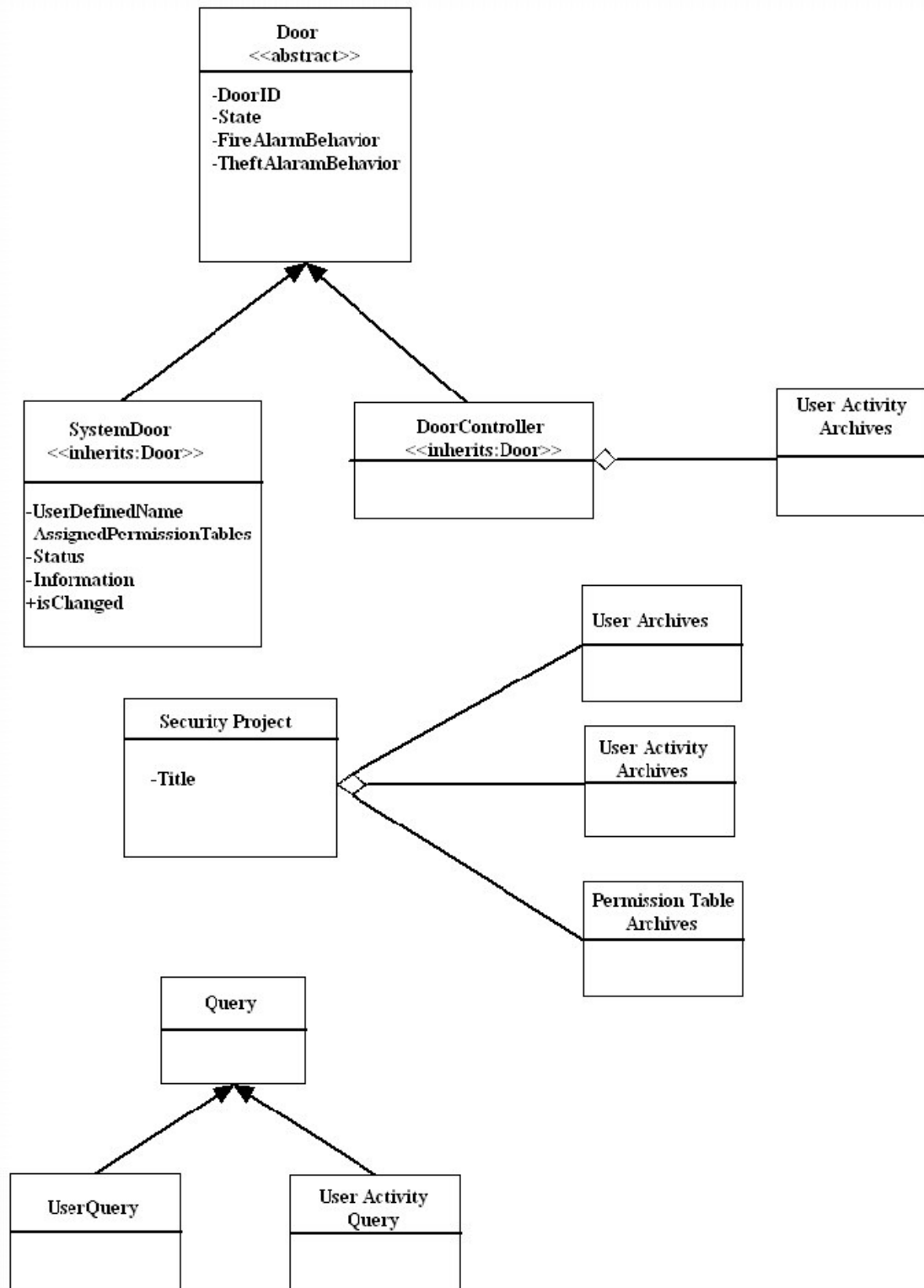


Figure-5: Inheritance and Aggregation Diagram for the Classes in the System

4.Database Model

Here, we provide a framework for the permanently stored data during the execution of system. We constructed a data model to show how the data stored, accessed and organized. Our model is mainly driven by the class diagram and the needs of application logic. In the following sub-sections we present this model to the developer using database tables, queries and file structures.

4.1.Database Design:

The part of the system which uses database management system for the information storage is the *Door Security Manager*. Therefore dbms is located at the master computer. Hereby, the data stored at the dbms is represented by describing the database tables. Moreover, there is a sub-section which describes query classes that contain parameters for a query and generate the appropriate SQL statement from these parameters.

4.1.1.Database Tables:

1)USERS: Stores all the registered users in the system with their attributes

```
CREATE TABLE USERS (  
userid int unsigned not null auto_increment,  
name varchar(20),  
surname varchar(20),  
address varchar(20),  
phone varchar(20),  
PRIMARY KEY(userid)
```

Query example: Fetches the name of the user whose userid is 12 from the USERS table.

```
SELECT U.name  
FROM USERS U  
WHERE U.userid = 12
```

2)PERMISSION: Stores the user membership to the permission tables for each permission table.

```
CREATE TABLE PERMISSION(  
tableid int,  
userid int,  
PRIMARY KEY(tableid,userid))
```

Query example: Fetches the userids of the users whom are the members of the permission table '12'.

```
SELECT userid  
FROM PERMISSION P1  
WHERE P1.tableid = "12"
```

3)PERMISSION_TABLE :Stores the attributes of the permission tables.

```
CREATE TABLE PERMISSION_TABLE (  
tableid int unsigned not null auto_increment,  
perm_title varchar(20),  
information varchar(100),  
PRIMARY KEY(tableid))
```

4)USER_ACTIVITIES: Stores the user activity.

```
CREATE TABLE USER_ACTIVITIES (  
userid int,  
action varchar(4),  
user_datetime date,  
doorid varchar(15),  
PRIMARY KEY (user_datetime, userid) )
```

Query example: Gives the user with userid and the doorid which he/she requested entrance at the specific time from the USER_ACTIVITIES table.

```
SELECT UAuserid, UA.doorid  
FROM USER_ACTIVITIES UA  
WHERE UA.user_datetime="12.24.2004.19.05"
```

Query example: Gives the names of the users who requested entrance.

```
SELECT U.name  
FROM USERS U, USER_ACTIVITIES UA  
WHERE U.userid=UA.userid
```

5)DOORS: Stores the attributes of all system doors

```
CREATE TABLE DOORS (  
doorid varchar(15) not null ,  
doorname varchar(20),  
doorinfo varchar(100),  
FireAlarmBehaviour varchar(),  
TheftAlarmBehaviour varchar(),  
IsChanged bool,  
state varchar(),  
PRIMARY KEY(doorid))
```

6) ASSIGNED: Stores information about the assignment between the tables and doors.

```
CREATE TABLE ASSIGNED (  
doorid varchar(15),  
tableid int,  
PRIMARY KEY (doorid, tableid),  
FOREIGN KEY (doorid) REFERENCES DOORS,  
FOREIGN KEY (tableid) REFERENCES PERMISSION))
```


4.1.2.Query Classes:

Query classes are classes that contain parameters for a query and generate the appropriate SQL statement from these parameters. Referring to the Figure-5 first we define the abstract class query and then the other two query classes.¹

	Nr	Name: Query Class Location: Door Security Manager Abstract Class
		Attributes:
#	1	Name: string Description: Name of the user, 20 characters length Default Value: “ “ which means a blank
#	2	Surname: string Description: Surname of the user, 20 characters length Default Value: “ “
#	3	UserID: integer Description: Unique ID of the user, stored in the card Default Value: -1
#	4	From: integer Description: userid of a user which is used to specify the first user while querying a group of users in the system. Default Value: None
#	5	To: integer Description: userid of a user, which is used to specify the last user while querying a group of users in the system. Default Value: None

¹ # represents protected members

	Nr	Name: UserQuery Class Description: Inherits from the Query class Location: Door Security Manager
		Attributes:
-	1	Name: string Description: Name of the user, 20 characters length Default Value: “ ” which means a blank
-	2	Surname: string Description: Surname of the user, 20 characters length Default Value: “ ”
-	3	UserID: integer Description: Unique ID of the user, stored in the card Default Value: -1
-	4	Address: string Description: Address of the user, 50 characters length Default Value: “ ”
-	5	Phone: string Description: Phone number of the user, 13 characters length Default Value: ” “
-	6	From: integer Description: userid of a user which is used to specify the first user while querying a group of users in the system. Default Value: None
-	7	To: integer Description: userid of a user which is used to specify the last user while querying a group of users in the system. Default Value: None
		Methods:
+	8	string CreateQuery() Description: Creates a query using its attributes and returns result as a string

Query example: Gives the users whose phone numbers begin with “0312”.

```
SELECT UserID
FROM UserQuery UQ
WHERE UQ.Phone="0312%"
```

Query example: Gives the names and surnames of the users whose surname begins with “K”.

```
SELECT Name, Surname
FROM UserQuery UQ
WHERE UQ.Surname="K%"
```

	Nr	Name: UserActivityQuery Class Description: Inherits from the Query class Location: master computer.
		Attributes:
-	1	UserID: integer Description: Unique ID of the user, stored in the card Default Value: -1
-	2	Name: string Description: Name of the user, 20 characters length Default Value: “ “ which means a blank
-	3	Surname: string Description: Surname of the user, 20 characters length Default Value: “ “
-	4	Action: string Description: It represents the pass or fail situation Default Value: “ “
-	5	DateTime: timestamp Description: Stores the date and time of the user entrance request Default value: None
-	6	DoorID: string Description: IP address of the embedded door controller Default Value: “-1.-1.-1.-1”
-	7	From: integer Description: userid of a user which is used to specify the first user while querying a group of users in the system. Default Value: None
-	8	To: integer Description: userid of a user which is used to specify the last user while querying a group of users in the system. Default Value: None
		Methods:
+	9	string CreateQuery() Description: Creates a query using its attributes and returns result as a string

Query example: Gives the userids from user activity archives who passed from the given door.

```
SELECT UserID
FROM UserActivityQuery UAQ
WHERE UAQ.Action="PASS" AND UAQ.DoorID= "32.18.01.01"
```

Query example: Gives the user activity of the users whose names are Tolga.

```
SELECT *
FROM UserActivityQuery UAQ
WHERE UAQ.Name= "Tolga"
```

4.2. Files

Files in the system are consists of the two part the header and the records.As expected the header part of the file gives information about the file and the records contain the data.

PERMISSION File		
Description: This file is stored in the door controller. Used by the door controller software to control user entrance permissions according to the State data located at the header.		
Header:	State	state
Record Structure:	userid	int

USERACTIVITY File		
Description: This file is stored in the door controller. Used by the door controller software to store user activity.		
Header:	doorid	varchar(15)
Record Structure:	userid	int
	action	varchar(4)
	user_datetime	date
	doorid	varchar(15)

5. Business Classes

In this section we will provide detailed information about business classes with their members and methods.All of the members and methods will be defined with their narrative explanation, and rigorous notation including their datatypes. But,before introducing business classes, we would like to introduce datatypes defined by us.

Custom Datatypes:

1) Name: state

Description: Represents the current working state of the door controller.

Valid Values: "OPENED", "CLOSED", "PERMISSIONTABLEASSIGNED"

2) Name: emergency

Description: Represents an emergency situation

Valid Values: "FIRE", "THEFT"

3) Name: action

Description: An action that can be applied to the door as a result of a user entrance request.

Valid Values: "PASS", "FAIL"

4) Name: behavior

Description: A behavior that can be set during an alarm situation.

Valid Values: "OPEN", "CLOSE"

5) Name: status

Description: shows the current status of the door controller

Valid Values: "OPENED", "PERMISSIONTABLEASSIGNED", "CLOSED", "TRANSFERSETTINGS", "TRANSFERARCHIVES"

We used following notation to represent classes:

	Nr	Name: Name Location: Location on the System Archthitecture
		Description:
		Members:
+/-	1	Member Name: datatype Description: Default Value: Leftmost column represents public private and protected property of the member.
		Methods:
+/-	1	Ret:datatype function (parameter:datatype); Description:

	Nr	Name: Security Project Location: Door-Security Manager
		Description: Contains and manages all settings of the wireless door security project.
		Members:
-	1	Title: string Description: Name of project, maximum 20 characters Default Value: "Default Project"
-	2	Users: UserArchives Description: Stores users registered to the the Project. Default Value: Empty
-	3	ActivityArchives: UserActivityArchives Description: Stores user activity records of the project Default Value: Empty
-	4	Permission Tables: PermissionTableArchives Description: Stores permission tables of the project Default Value: Empty
-	5	Door Network: NetworkHandler Description: Handles communication between the security project and embedded controllers. Default Value: None
-	6	Doors: SystemDoor[] Description: An array that holds the doors of the system Default Value: Empty
		Methods:
+	1	bool OpenFromDatabase () Description: Loads the security project data from the database , and returns true if the operation is successful.
+	2	SystemDoor FetchDoor (DoorID: string) Description: Returns the door object with the specified ID.

+	3	<code>bool AddUserToProject (NewUser: User)</code> Description: Adds a new user to the User's table in the database
+	4	<code>bool RemoveUsersFromTheProject(Query: UserQuery)</code> Description: Removes users from the User's Table according to the specified Query object and returns true if the operation is successful.
+	5	<code>bool EditUserInTheProject(UserID: integer, Name: string, Surname: string, Phone: string, Address: string)</code> Description: Edits the user information of the user with the specified UserID and returns true if the operation is successful.
+	6	<code>bool UpdateDoor(Door: SystemDoor)</code> Description: Updates the door controller according to the door object and returns true if the operation is successful.
+	7	<code>bool AddPermissionTableToTheProject(Title: string, Information: string)</code> Description: Adds a new permission table with the specified Title and Information to the database and returns true if successful.
+	8	<code>bool DeletePermissionTableFromTheProject (TableID: integer)</code> Description: Deletes the permission table with the specified TableID from the database, and returns true if the operation is successful
+	9	<code>PermissionTable GetPermissionTable(TableID: integer)</code> Description: Returns the permission table with the specified ID.
+	10	<code>bool EditPermissionTable(TableID: integer, Title: string, Information: string)</code> Description: Edits permission table attributes according to Title and Information parameters and returns true if the operation is successful.
+	11	<code>void AssignTableToDoor (TableID: integer, DoorID: string)</code> Description: Assigns a permission table to a door in the project.
+	12	<code>bool AddUsersToPermissionTable (Query: UserQuery, TableID: integer)</code> Description: Adds users represented with the specified UserQuery object to the permission table specified by the TableID and returns true if the operation is successful.

+	13	<p><code>bool RemoveUsersFromPermissionTable(Query: UserQuery, TableID: integer)</code></p> <p>Description: Removes the users specified by the UserQuery object from the permission table specified by the TableId, and returns true if the operation is successful.</p>
+	14	<p><code>bool UpdateProjectArchives()</code></p> <p>Description: Updates the project archives by fetching data from the door controllers and storing it in the user's activity table..</p>
+	15	<p><code>bool GetUserActivity(Door: SystemDoor)</code></p> <p>Description: Fetches the stored user activity from the door controller pertaining to the Door object given as an input parameter and returns true if the operation is successful.</p>
+	16	<p><code>bool ClearProjectArchives(Query: UserActivityQuery)</code></p> <p>Description: Deletes the user activities specified by the UserActivityQuery object from the user's activity table.</p>
+	17	<p><code>UserActivityTable QueryActivity(Query: UserActivityQuery)</code></p> <p>Description: Queries user activity table with the Query object and returns the result as a UserActivityTable.</p>
+	18	<p><code>bool UpdateDoorStatus(Door: SystemDoor)</code></p> <p>Description: Updates the remaining storage of the door controller according to the given Door object and returns true if the operation is successful.</p>
+	19	<p><code>void ClearAssignment(TableID: integer, DoorID: integer)</code></p> <p>Description: clears the assignment of a permission table, door pair with the specified TableID and DoorID.</p>

	Nr	<p>Name: SystemSecurity</p> <p>Location: Door-Security Manager</p> <p>Description: Security class that handles authentication</p>
		Members:
-	1	<p>Password: string</p> <p>Descriptions: Password of the project in string format, 6-12 characters in length</p> <p>DefaultValue: Developer Determined</p>

		Methods:
+	1	bool CheckPassword(passwd: string) Description: Validates the password input during login, returns true if the password is equal to Password.
+	2	string GetPassword() Description: Returns Password
+	3	bool ChangePassword(OldPassword: string, NewPassword: string) Description: Checks whether the old password equals to Password member, if they are equal it changes Password member to NewPassword.
+	4	void SetPassword(NewPassword: string) Description: sets NewPassword as Password.

		Name: Door
	Nr	Description: Abstract class that provides base for the SystemDoor and DoorController classes.
		Members
-	1	DoorID: string Description: IP address of the embedded door controller. DefaultValue: “ -1.-1.-1.-1 ”
-	2	State: state Description: Represents the state of the door controller DefaultValue: “OPENED”
-	3	FireAlarmBehavior: behavior Description: Type of behavior system will be set in a fire situation. Default Value: “OPEN“
-	4	TheftAlarmBehavior: behavior Description: Type of behavior system will be set in a theft situation. Default Value: “CLOSE“

		Name: SystemDoor Location: Door-Security Manager
	Nr	Description: System door object that represents a door-controller in Door Security Manager. Inherits From: Door Class
		Members:
-	1	User Defined Name: string Description: User defined name of the door Default Value: Door ID in string format
-	2	Assigned Permission Tables: integer [] Description: An integer array that holds the ID's of the tables assigned to the door. Default Value: None
-	3	Information: string Description : User comments about the door Default Value: “ “
+	4	IsChanged: bool Description: A boolean value which is changed to true after user operation to indicate change in object. Default Value: false
-	5	Status: integer Description: An integer value between 0 to 100, which represents the fullness of the door's disk. Updated by master computer. Default Value: None
		Methods:
+	1	void ChangeState(New State, State) Description: Changes state of the door to the New State. Sets IsChanged attribute to true. Also saves changes to the door object representation on disk.
+	2	void EditDoor(New User Defined Name: string, state:state, Information: string, FireAlarmBehaviour:behaviour, TheftAlarmBehaviour: behavior) Description: Edits the attributes of the Door object according to the function parameters. Sets isChanged attribute to true. Also saves changes to the door object representation at the disk.

+	3	void AddToTableList (Table ID) Description: Adds the permission table with the specified ID to the door object and it's representation on the disk. And sets isChanged attribute to true.
+	4	void SetStatus (Percentege: integer) Description: Sets the value of member Status the argument Percentage. The value of isChanged member is also set to true for the specified object.
+	5	void RemoveFromTableList (TableID) Description: Removes the permission table with the specified ID from the door object and its representation on the disk. Also sets isChanged attribute to true.

	Nr	Name: DoorController Location: Door-Controller Software Description: Door Controller representation in the embedded system software Inherits From: Door Class
		Members:
-	1	Local Archives: User Activity Archives Description: Archive object that stores the user activity of that door Default Value: None
		Methods:
+	1	void SignalEmergency (EmergencyType: emergency) Description: An interrupt function that signals emergency, and changes door behavior according to emergency type.
+	2	void SetDoorBehavior (Behavior: behavior) Description: Set door behavior, according to given Behavior parameter.
+	3	action CheckID (UserID: integer) Description: Checks UserId from the permission table, and returns an Action. Also it stores the Action as a User Activity in Local Archives before the function returns.
+	4	bool CheckPermissionTable (UserID: integer) Description: Checks Permission Table for the specified UserId and returns true if the UserID found in the Table

	Nr	Name: User Location: Door-Security Manager
		Description: Registered user in the project, card owner
		Members
-	1	Name: string Description: Name of the user, 20 characters length Default Value: “ “
-	2	Surname: string Description: Surname of the user, 20 characters length DefaultValue: “ “
-	3	UserID: integer Description: Unique ID of the user, stored in the card. DefaultValue: -1
-	4	Address: string Description: Address of the user, 50 characters length.
-	5	Phone: string Description: Phone number of the user, 13 characters length.

	Nr	Name: UserArchives Location: Door Security Manager
		Description: Archive object that manages the users stored in a data store.
		Methods:
+	1	<code>bool Add(NewUser: User)</code> Description: Add new user to the User's table. Returns true if the operation is successful.
+	2	<code>bool Edit(UserId: integer, Name: string, Surname: string, Phone: string, Adress:string)</code> Description: Edits the attributes of the user with the specified ID and returns true if the operation is successful.
+	3	<code>User[] GetUsers(Query: UserQuery)</code> Description: Returns the Users specified by the Query object as a User array from the User's table.

	Nr	Name: UserActivity Location: Door-Security Manager and Door-Controller Software Description: It is a record which keeps the information about a specific user entrance request.
		Members
-	1	UserId: integer Description: The Id of a user which is given with the card DefaultValue: “ -1 ”
-	2	Action: action Description: It represents the pass or fail situation. DefaultValue: None
-	3	DateTime: timestamp Description: Stores the date and time of the user entrance request DefaultValue: None
-	4	DoorID: string Description: IP address of the embedded door controller. DefaultValue: “ -1.-1.-1.-1 ”

	Nr	Name: UserActivityTable Location: Door Security Manager and Door-Controller Software Description: Table of users
		Members
-	1	Title: string Description: Stores the title of the user activity table. Default Value: “ Untitled “
-	2	DateTime: timestamp Description: Stores the date and time of the user table was last modified. Default Value: “NONE”
-	3	activities: UserActivity[] Description: Table storing user activities of the particular table Default Value: “ “

	Nr	Name: UserActivityArchives
		Location: Door Security Manager and Door-Controller Software
		Description: Archive class that manages the user activity stored in data store
		Methods:
+	1	bool Add(NewUserActivityTable: User Activity Table) Description: Add NewUserActivityTable to the UserActivityArchives of the project.
+	2	bool ClearArchives(Query: UserActivityQuery) Description: Deletes the User Activity Records specified by the UserActivityQuery object from the UserActivity table, and returns true when the operation is successful.
+	3	UserActivityTable QueryUserActivity(Query:UserActivity Query) Description: Queries the user activity archives with the Query object and returns the result as a UserActivityTable.
+	4	void StoreUserActivity(UserId: integer, Action: action,DateTime: TimeStamp) Description: Stores user activity into the LocalArchives. It takes the ID, Action and DateTime parameters and turns them into a UserActivity and add it to the LocalArchives.

	Nr	Name: PermissionRecord
		Location: Door-Security Manager and Door-Controller Software
		Description: Contains UserID of the permitted user
		Members
-	1	UserId: integer Description: Unique number which is assigned to each user. DefaultValue: “ -1 “

	Nr	Name: PermissionTable Location: Door-Security Manager, Door-Controller Software
		Description: Table that holds the permission records for the permitted users.
		Members:
-	1	Title: string Description: Name of the permission table. Default Value: "Untitled"
-	2	TableID: integer Description: Unique identification number of the permission table. Default Value: Auto generated
-	3	Information: string Description: User comments about the permission table.

	Nr	Name: PermissionTableArchives Location: Door-Security Manager
		Description: Archive object that manages the permission tables stored in the data store.
		Methods:
+	1	PermissionTable MergeTables(PermissionTables: integer[]) Description: Merges tables with the specified ID's in the permission tables' integer array and returns the merged table.
+	2	bool Add(Title: string, Information: string) Description: Adds the permission table with the specified Title and Information to the data store and returns true if operation is successful.
+	3	bool Delete(TableID: integer) Description: deletes the permission table with the specified ID and returns true if the operation is successful.
+	4	PermissionTable Get(TableID: integer) Description: returns the permission table with the specified ID.
+	5	bool Edit(TableID: integer , Title: string , Information:string) Description: Edits the table attributes according to Title and Information parameters and returns true if the operation is successful.

+	6	bool AddUsersToTable (TableID: integer, Users: User[]) Description: Adds Users to the permission table specified by TableID and returns true after successful operation.
+	7	bool RemoveUsersFromTable(TableID: integer, Users: User[]) Description: Removes the users contained in Users from the permission table specified by TableID.

	Nr	Name: Card Location: Outside world Description: Physical object, UserID is embedded in it.
		Members
-	1	UserID: integer Description: Unique number which is assigned to each user. DefaultValue: “ -1 “

	Nr	Name: CardReader Location: Outside world Description: Is the physical device belong to system which is responsible for reading cards
		Methods:
+	1	action PassCard(Card: card) Description: It extracts the UserID from the user’s card and sends it to the door controller for verification. The door controller then returns the appropriate action to be applied by the card reader.

	Nr	Name: Electronic Lock Location: Is located on the door Description: Is the physical device that controls the opening of the door
		Methods:
+	1	<code>action Apply(Action: action)</code> Description: Applies the system's action to the door.

	Nr	Name: StatusIndicator Location: Door Controller Hardware Description: An output device which indicates the current status of the door controller.
		Members:
-	1	Status: status Description: current status of the door controller represented by the output device. Default value: "Waiting"
		Methods:
+	1	<code>setStatus(Newstatus: status)</code> Description: Sets the current status of the StatusIndicator to the NewStatus.

6. Door Controller Software

The main function of the interface circuit is to convert the wiegand data format to RS-232 data format. The circuit also checks the data it receives from the card reader for any errors i.e. by checking the parity bits. We implemented an error checking facility as follows:

- The circuit will check the card ID data it receives for an error, if an error is detected the card ID data is ignored, if the data is error free, the card ID data will be send to the serial port of the door controller in RS-232 data format.

The interface circuit will consist mainly of a PIC microcontroller and the necessary peripheral circuitry. The PIC microcontroller will be programmed to perform the data conversion from wiegand format to RS-232. The pseudocode of the PIC software is as shown below:

Main Program (Endless Loop):

```
While( 1 ){  
If( Number Of Bits Read == 26 )  
    TransmitDataToSerialPort();  
If( Timer-Out == True )  
    Number Of Bits Read = 0  
}
```

Interrupt service routine:

```
Read wiegand inputs into port register  
Extract Data_0 and Data_1 lines  
Check if they are both low      /* error both lines cant be low */  
Exit Function if both lines are low  
Check if they are both high     /* error both lines cant be high */  
Exit if both lines are high  
Read wiegand bit                /* bit = 0 when Data_0 = 0  
                                bit = 1 when Data_1 = 0 */  
Increment the number of bits read  
Load Time-Out Timer
```

TransmitDataToSerialPort:

```
    Disable Interrupt pin  
    Initialize bit counter to 8  
    Initialize number-of-bytes-sent to 3
```

SendByte:

```
    Get byte to be sent  
    Send start bit
```

Loop: Rotate the transmit byte right (into carry)

```
    Move carry bit to serial output pin  
    Wait one bit time  
    Decrement the bit counter  
    Is the bit-counter zero?  
        No, jump to Loop:  
        Yes, Stop:
```

Stop: Send stop bit
Is the number-of-bytes-sent zero?
No, jump to SendByte
Yes, Exit:

Exit: Enable interrupt pin

Below is a state diagram showing the different states of the Interface Circuit:

