MAINTENANCE SERVICE REQUEST PROCESS OPTIMIZATION
FOR THE CLIFFS RESORT AT SHELL BEACH

by

JOHN BIRCH

A Senior Project submitted
in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Industrial Engineering

California Polytechnic State University
San Luis Obispo

Graded By: _____________________ Date of Submission: _____________________
Checked by: _________________ Approved By: _____________________
Abstract

This process optimization project utilizes a new idea to have an Access 2007 database automatically send a text message to a given cellular telephone. The hotel’s maintenance service request process is outdated and inefficient, relying solely on paper and human memory for success. Initiation, tracking, and recording the maintenance service requests need to be automated to optimize this process. The solution is a working database which employs said automation using Access 2007 in combination with a separate program created to facilitate the automatic text messaging. Implementing this database will result in higher customer satisfaction for the hotel.
# Contents

Abstract .................................................................................................................. 2
Introduction .............................................................................................................. 5
Background ............................................................................................................. 7
Literature Review .................................................................................................. 8
Design & Methodology .......................................................................................... 17
Features .................................................................................................................. 21
  Predefined Compiled List of Rooms and Common Problems ........................................... 21
  Automatic Text Messaging .......................................................................................... 22
  Logging and Tracking Service Requests ..................................................................... 24
  Adding New Types of Common Issues ........................................................................ 24
  Call-back Timer ......................................................................................................... 25
  Counting Recurring Issues ......................................................................................... 25
  Tracking Unresolved Requests .................................................................................... 26
  Organized Display of Time Elapsed ........................................................................... 28
  Unresolved Requests Subform ..................................................................................... 31
  Conditional Formatting – Red and Bold Issues Older Than 15 Minutes ...................... 33
  Call-back Reminder Added Functionality ..................................................................... 33
  One Room, Multiple Requests .................................................................................... 34
  The Form has Become the Database .......................................................................... 35
Results and Discussion .......................................................................................... 36
Conclusion and Future Improvements ....................................................................... 38
  Conclusion .................................................................................................................. 38
  Future Improvements ................................................................................................. 38
Acknowledgements .................................................................................................. 40
Appendix .................................................................................................................... 41
  Version 1.0 .................................................................................................................. 41
  Screenshots ................................................................................................................ 41
  Coding ........................................................................................................................ 41
Introduction

This project will delve into the inner workings of The Cliffs Resort at Shell Beach and optimize a quite non-optimal process. The process to be analyzed will be their maintenance service calls. After working there for a year, I noticed that many ideals from the industrial engineering field could be applied to the business, resulting in great improvements.

The idea to work on this particular process originated from a meeting with the Rooms Executive and the General Manager of the hotel. It was determined from this meeting that improving a maintenance process would have the greatest effect on customer satisfaction at the hotel. At that point, a few of their maintenance processes were discussed. The instant their service request process was shown to use paper and radios, I realized there would be a way to eliminate both of those using some clever automation.

Two problems were addressed when creating this project. One is that many service calls are not being completed at the hotel, due to the inefficient format of the hotel’s current service request process. Front desk agents are forgetting to call the maintenance employee, or the requests are communicated unclearly, etc…. Another problem is that they are currently logging all of these requests on paper, producing poor track-ability and logging. Hotel executives should have a clearly printed document showing the most prominent issues in the rooms when making decisions for improvements.

To optimize the process of a maintenance service call, the hotel’s normal procedures were closely examined. From this examination, it was determined that a database needed to be designed to make the process mostly electronic, taking away the pen and paper. This database would also remove the need for an employee to physically call the maintenance worker. This will
take some research in computer programming to give the database the capability to send text messages.

This project will deliver a working database, completely customized for The Cliffs Resort, that will have the capabilities to log all incoming data about issues within the hotel, output charts and information about most common issues, automatically send text messages to active cell phones, and remind the employees at the front desk to follow-up with guests.

In order to create the database with these capabilities, much research involving computer programming was needed. The particular programming languages used in the project are VBA (Visual Basic) and C#.

The next section of the report will go into the background of the hotel, followed by a literature review of similar projects and research focused on process optimization within the hotel industry. The report will then describe the design and methodology involved with creating the solution to the problems discussed above. Finally, results from implementing this solution will be discussed followed by a conclusion.
**Background**

The Cliffs Resort at Shell Beach has been in existence for over 20 years. The hotel has changed ownership at least three times since the hotel was erect in the 1980’s. The most recent owners, King Ventures, have been the most generous of all the owners. They are generous in the sense that they spend a very high percentage of the profit they make from the hotel on improvements, salaries, and maintenance of the establishment. The result of this generosity has been a very successful business even in this recent time of recession.

Only recently has the hotel begun to attempt to “get with the times,” as my generation would say. Less than a year ago, they upgraded the televisions in all of the rooms from standard definition, tube TV’s to high definition, flat panel LCD TV’s. At the same time, they implemented a streaming high definition video system as an upgrade to their room full of VCR’s and VHS tapes that guests were using to order movies.

The age of the hotel has been a significant factor in their success, having made such a great, lasting name for themselves. However, their old-fashioned ways and inefficient habits are beginning to come to light as painfully costly practices that need to be refined.
Literature Review

The focus of this review will be on methods used to optimize multiple processes that could be applied to the hotel industry. One process of interest is maintenance and housekeeping service requests. This review will examine methods that are in place and evaluate not only weaknesses that can be eliminated, but also the strengths that can be applied towards a senior project at The Cliffs Resort.

A journal on human resources in the hotel industry reveals an impact of employee training on both job satisfaction and intention of employees to stay. The journal suggests that training in this industry in particular is always in need of updating and improvement. This research was based on the performance metric known as “expectations.” It’s defined as desired performance levels, which reflect an attribute’s importance to a person. These expectations are directed towards the hotels service quality in five dimensions: reliability, responsiveness, assurance, empathy, and tangibles. (Chiang 100-101)

The journal had many hypotheses that were tested. These included tests for management perceptions of employee expectations of training quality being different than employee expectations. They also tested for training quality being related with an intension to stay. They collected data from hotels in Kansas and Missouri. The criteria for the hotels was to be members of the American Hotel and Lodging Association state affiliate, and the employees must have received some type of formal or informal training. They then used surveys to collect the data, which were returned to them by the human resources directors. (Chiang 104)

They found that there was a gap between employee expectations and perceptions of the training quality, which indicates that employees are not satisfied with the training quality. The desired
outcome of the study was to encourage hotels to give training a higher priority. They were able to show training quality influences an employee’s intention to stay through job satisfaction. (Chiang 117)

This study was helpful to show that employees prefer higher quality training. Having a metric like job satisfaction and willingness to stay is good because it can be shown in monetary form. If a company puts more time and energy into training an employee, it will pay off in the end because the employee is willing to stay longer. In addition, a positive attitude from job satisfaction in the service industry is great for return customers.

Some articles and online sources pointed to an incredibly useful program for hotels called GuestWare will be examined. Guestware 3.0 was designed to completely take over the way hotels handle service calls. Nearly everything becomes automated. Typically, someone at the front desk of a hotel will answer the phone when a guest calls. It’s then their responsibility to write down the room and problem, contact the maintenance employee on staff, and remember to call the guest some time later to make sure the issue has been resolved. Nothing is logged, so hotels are unable to analyze the most common problems without guessing and relying on the memory of the front desk employees.

Once Guestware is installed, everything becomes incredibly simple. Whoever answers the call only needs to enter the room number and issue. It will have a “quick search” type dropdown box for the most common issues for faster input. Once the employee enters this information, a service call is automatically generated. The program has the capability to automatically send text messages to the staffed maintenance employees. These employees have to confirm the call or
else the program will notify management that the issue is not being resolved. This program also has the capability to prioritize:

“GuestWare's Work Orders system provides the ability to prioritize and monitor necessary maintenance tasks to ensure the highest level of quality. It also helps improve work flow by providing a systematic way to track and analyze work orders. GuestWare’s Work Orders system provides management with the tools it needs to better manage labor and product quality control.” <Guestware.com>

Along with all this to help the guests get fast and effective service, Guestware includes the ability to track many important aspects. Preventative maintenance printouts include the most recent service call issues, as well as the top 5 most common within 18 months. It will also track response time and costs incurred from each work order. <Guestware.com>

According to a case study done at Tropicana Resort and Casino in Las Vegas, Guestware made a significant improvement to their customer service, and they didn’t even fully implement the program! They have the work order generator and tracking program, but they decided not to take advantage of the two-way text messaging capabilities. <Guestware.com>

Guestware is simply an amazing idea when applied to certain situations. It is able to keep thousands of people on the same page, while keeping response times for customer service extremely low. The system’s two-way text messaging capabilities are impressive and necessary for a hotel to be able to appreciate all of the benefits the software has to offer.

However, Guestware requires expensive training and people that are ready to change. There are at least 14 different courses the software creators offer that average to about $250 a person. One also would have to learn something new and break quite a few old habits. If the employees are willing to learn and the employers are willing to spend, the software and training are a great acquisition.
An article from Hotel & Motel Management compares the technology used by Guestware to The Jetsons, a futuristic cartoon. The article is based on using the latest technology to enhance a hotel’s housekeeping capabilities. It discusses using tracking chips for In-Room-Dining trays to alert staff when it’s outside the room and ready to be picked up. A large section of the article discusses a hotel that implemented Guestware and benefitted greatly from it. The article finishes by discussing a hotel using the latest of technology to increase efficiencies. They use a program that takes advantage of the iPod touch, which is basically a pocket-sized laptop with WiFi capabilities and applications. That technology would streamline the entire housekeeping staff, allowing a continuous network to be maintained. (Crowell)

The chips to track the trays are great for hotels with inset doors, because employees might not walk all the way down the hall when looking for trays that are ready to be picked up. The trays would be hidden from sight when looking down the hallway. However, the cost and monitoring of these chips would be quite expensive and probably not worth the investment for smaller resorts. Taking advantage of the iPod touch’s WiFi capabilities to improve any process would be profitable in nearly any application that comes to mind. Even smaller resorts would benefit because the cost is simply related to the size of company.

An article produced from the same magazine in November of 2008 stresses the effectiveness of a room’s scent and lighting. A displeasing scent, even if it’s just in the back of the guest’s mind, will always leave a negative impact on their opinion of the hotel. The article advises to always check for mold, as it is a common issue in the industry. A preventative maintenance strategy is to keep the rooms warmer when they’re not occupied, as mold typically grows under certain conditions. The article also recommends using CFL light bulbs in place of the traditional incandescent for energy saving purposes. However, the article warns to be careful of the color
temperature of the new bulbs. It is important to maintain the ambiance originally intended by the
designers. (Feingold)

This brings up a great point for preventative maintenance. Hotels should definitely have routine
checks for mold if it’s a common problem in the industry. Energy conservation is also great, as
long as customer satisfaction, and in turn profit, does not take a huge dive.

An article in Lodging Hospitality introduced some new guestroom lock offerings. These new
locks have two main purposes. One is to meet the needs of disabled guests with new handle
designs and exterior hardware. The other is to meet the requirements and interoperability
standards in order to help vendors and hoteliers “speak the same technological language.” The
locks are designed by Kaba with ATLAS (advanced technology lodging access system). The
relatively new ATLAS branch will be the driving force in these lock designs. Sony will also be
supporting a new lock system that uses high security smart chip cards that can be used for
different functions within a commercial environment. (Technology Today)

Providing accommodation for disabled guests is not only optimal because it meets requirements,
but it is also helpful for business. Things that typically help out a disabled guest can also be
considered more luxurious for a guest that is not disabled. Bigger bathrooms and automatic
doors, for example, are easier to use, and even preferred by a guest that appreciates the luxury
included. This way, if a disabled person has not reserved the room, they can even be used as
junior suites.

Hotel & Motel Management released an article in 2004 that encouraged the use of wireless
applications in the hotel environment. The article, being five years old, made the point that
wireless high speed internet was becoming more and more prevalent. It discussed the use of
wireless applications to improve operations and guest services. A helpful newly designed piece of technology discussed in the article is quite similar to a point-of-sale device used in retail stores. This device could be used by a waitress to take an order and wirelessly transmit the order to the kitchen. This would allow waitresses to stay out on the floor, and even help keep track of orders. (Adams)

The amazing thing about all of these innovative technologies is really the “bonus” that comes with all of it. They’re designed to help do things more effectively by using electronics and such, which by itself is awesome. In addition, the incredible bonus that comes with all of these electronics is the ability to automatically log everything that’s happening. It requires little to no effort to output everything that goes into these electronic devices into a spreadsheet or chart to analyze and further improve any business that’s using them.

A paper discussing innovations in the hotel industry from Science Direct determined that innovations in general have a positive impact on the hotel performance. This paper is super technical and incredibly boring. Probably got an “A” for effort and nobody read it. (Mattsson)

An article was published in 2007 about sending mass text messages as notifications in emergency situations. The idea that text messaging is the most efficient way of reaching individuals is resounded by this article. This project is primarily based on that concept – eliminate streaming two-way communication and enable efficient text alerts. The system described by this article also has the ability to receive feedback texts. This capability is spectacular for updating the status for any item in a database. The primary use of the system is to alert a mass of people about emergencies or missing persons, etc. However, the basic concept is
something that will soon be an integral part of our society as a necessary form of communication. (Spadanuta)

Wells Fargo presented a new service in May of 2010 called “Rapid Alerts.” This will be a free service for their customers who can use it to track their spending by having event-triggered text messages sent to their cell phones based on pre-determined types of activity or transactions with their accounts. Again, companies are starting to realize the best and most efficient way to communicate with people in today’s society is through text messaging. The technology for this cutting edge service is provided by VisaNet, Visa’s global processing network. Wells Fargo sends the alerts, VisaNet delivers them. “Rapid Alert messages contain the amount, time and date of the transaction, as well as currency conversion and information relating to the merchant.” (PR Newswire)

In March of 2009, AlarmSoft developed a new capability for their central station software. The new technology automatically sends text messages triggered by events which contain account information. The recipients of the text messages and the information sent are all predetermined in the software. They named this addition “Signal-Me.” This software was designed to eliminate the need for a dispatcher to manually send out alerts based on the alarms going off. This particular function is very similar to this senior project in the sense that the database becomes the new “dispatcher” to summon the service of the maintenance employee and deliver the required information. (Anonymous)

A recent article from Hotel Business on sustainability exclaims that sustainability is about much more than “being green.” The article notes that many companies in the hospitality industry are slow to embrace the ideals of sustainability. What typically happens is even if a hotel seems to be
all about sustainability, many of their actions will contradict what they claim to believe. For example a hotel that advertises that they are “green” may not have recycling receptacles.

“Companies must take an integrated approach when developing strategies to ensure their business can sustain themselves economically and sustain the resources they require, both in terms of people and the environment. For example, an environmentally friendly business is great in theory but only if it can make money to sustain itself and continue to do its good work. A profitable business that depletes all of its resources or loses favor with its customers or employees can also result in failure.” (Connolly)

If a hotel decides to embrace sustainability, it has to be for the right reasons. If they do it to simply gain business, they will likely miss something and not be sustainable in some areas. This negligence will reveal their phoniness and end up costing them in the long run. Even hotels that choose to be sustainable to truly be eco-friendly, they might get caught forgetting to sustain a cash flow. This article makes a great point that choosing sustainability can be a great move, as long as it’s done carefully.

In conclusion, the findings of these studies and suggestions in these articles have many benefits, but also have some downfalls. The Guestware program was very well developed, and will offer much great advice for designing a senior project. However, that particular program was designed for much larger hotels (1,700+ rooms) while the hotel this project is intended for has only 160 rooms. A customized, easy to learn program will be much more helpful to this smaller company.

The study on training quality being related to job satisfaction gives essential advice for this project. It shows that the training for the implemented program should be high priority in order for it to actually be helpful.
Applying the latest technology to this senior project will be extremely effective. Using the iPod Touch and cell phone text messages to maintain a streaming network will be one of the main reasons for this project’s success.

Taking a look at the hotel’s accommodations for disabled guests may prove helpful, however this will not likely be the project’s main focus. New locks and handle designs might be too expensive to be a feasible improvement for the hotel at this point in time. Sustainability would fall in the same boat here. It will at least be briefly examined in the project, but it is not anywhere near the main focus.

All in all, reading and evaluating these articles will be exceptionally useful for rendering a successful senior project. Improving on methods and taking advantage of the latest in technology is simply how we progress as a race.
Design & Methodology

The solution for this problem started as a spark in my mind when the process was first described to me. A front line manager described that the process was done completely on paper and relied on the memory of the front desk agents for completion of these service requests. Currently, a guest will call down with an issue in the room, and the front desk agent writes down the room number, what the issue is, and what time the guest called. Next, they need to radio the maintenance employee to convey this information. Finally, they need to remember to call the guest back 15 minutes later and follow up to find out if the issue has been resolved.

Almost instantly after this process was described, I knew not only that it was incredibly inefficient, but that this could all be solved using some kind of program. As I thought more about what kind of program to use, I knew an Access database should have all the capabilities necessary to at least take the paper out of the equation, as well as the reminder to call the guest back. I imagined there might even be some way to have this database convey information to the maintenance employee instead of having to radio them.

Once I hit this point in my thought process, I realized there was something I have never seen an Access database do that, if I could figure out how to do it, would be exactly what I needed to fully optimize this process. I wanted to have the database send a text message triggered by the click of a button. This thought became what I built the entire project upon. I knew that if I could create a database that could send text messages, everything else would be relatively easy to design.
Flowcharts

There are really two different processes that need to be charted out for the front desk agent interacting with the database. The first is initiating the service request. This process begins when the guest calls down with the issue and ends when the issue becomes a service request in the system. The second process is initiated by a pop-up window reminding the front desk agent to call the room and find out if the issue has been resolved. This process ends with the issue becoming resolved in the system (on rare occasion the issue will remain in the system if the maintenance employee is unable to resolve the issue).
A Guest has a maintenance issue in Room XYZ
The front desk agent picks up the phone and begins conversation.

Is the Maintenance Request Management System (MRMS) Open?

Go to Desktop and open MRMS

(Make sure window is active/ maximized)
Enter Room XYZ in Room box

Find out what the problem is with the room

Is this problem already in the pre-defined list?
(Click the dropdown box labeled “Issue”)

Select the applicable issue

Manually Type the guest’s issue into the box

Click “Initiate Request”
(A Text Message will automatically be sent to the maintenance employee)

Inform the guest that a maintenance employee will be there to assist them shortly

Figure 1 - Flow chart of request initiation
A pop-up window just came up on your screen telling you to call Room XYZ and find out if the issue has been resolved.

Don’t click Yes or No yet!

Look over the list and find what the issue is with the corresponding room.

Call the room. Has the maintenance employee resolved the issue?

Click “Yes” on the pop-up window.
The issue will disappear from the list and become logged as resolved.

Call the room after at least X minutes have past to follow up and ensure the issue has been resolved.

“Tam busy and haven’t begun to resolve this issue yet. Give me X minutes.”

Has the issue been resolved?

Yes

Click the corresponding “Resolved” checkbox for the issue.
The issue will disappear from the list and become logged as resolved.

Radio the maintenance employee.

“I don’t have the necessary resources to resolve this issue”

Inform your supervisor that this issue requires special attention and the maintenance employee is unable to resolve it.

No

“Click “No” on the pop-up window and radio the maintenance employee. Make sure he’s handling the situation.

The issue in the list will now turn red and bold faced, meaning it has been unresolved for longer than 15 minutes. This means it will require your close attention to ensure the issue gets resolved

What was the maintenance employee’s response when you radioed him?

Has the issue been resolved?

Yes

No

Radio the maintenance employee.

Figure 2 - Flow chart of request resolution
Features

Predefined Compiled List of Rooms and Common Problems

I started by creating a “Rooms” table in the database, which is simply a list of every single room in the hotel. This was necessary because the form would require the front desk agent to input the room that had the issue, and I wanted the database to have the integrity that would not allow them to input an incorrect room. Hopefully, this would catch them when they might otherwise accidentally tell the maintenance employee to repair the toilet in room 5105 (a room that doesn’t exist at The Cliffs). I also created a table called “Common Problems” which would be a compiled list of common problems that are currently known in the hotel. The few I added initially were Television, Air Conditioner, and Toilet. Neither of these two tables needed to have any kind of autonumber ID’s, simply because these were lists that I would need to refer to directly by the information contained within each cell. Finally, I created a form called “Maintenance Service Request” that had combo boxes referring to these two tables. This was the first version of the database I would want to create as a solution for this project. The design of the database continued through creating additional versions as I added new capabilities. The initial tables and forms I created are displayed below:
Automatic Text Messaging

At this point, I needed to give my database the capability to send a text message. This began a journey of learning and research spanning a few weeks that resulted in a button that could send a text message to whatever cell phone I desired, containing whatever was in the boxes on the form after “Room” and “Common Problems.”

The journey began with consulting a close friend, named Chris Mathews. This computer programming expert informed me that I would need access to an SMS server in order to send text messages automatically; otherwise they would be discarded as spam. He then directed me to a website explaining a way to send text messages using email. For example, you could text my cell phone my emailing “9163468279@vtext.com”. At this point, we began creating a separate executable program that could interact with an SMS server I had access to. Using Microsoft’s Visual C#, we created a program called SMS_Console.exe. This program requires 4 arguments in order to function: To, From, Subject, and Body. If all arguments are provided, the program will use my SMS server to send a text message to the “To” address which includes the three remaining arguments.

Now, all I needed was a seamless way to give this program these arguments. This is where knowledge of VBA coding came in handy. The room and issue would be in the body of the message, and the subject would simply be “Service Request.” The initial versions of the database used “service@cliffsresort.com” as the given sender in the code, this was to fulfill the need for the “From” argument. For now, the “To” argument is determined in the code, invisible to the operator of the form.
In order to make this all work in VBA, I defined process, room, item, recipient, sender, subject, and body as strings that would interact with SMS_Console.exe to send the desired text message including all the required information. This code was all written under the On Click function of Command14, which was the button the user would click once they had finished filling out the form.

I directed the room and item strings to the two combo boxes described above. The recipient, sender, and subject strings are pre-defined within the VBA code. The recipient has been coded to be my cell phone number for testing purposes, but the finished project will have the maintenance employee’s cell phone number. I set the sender to “service@cliffsresort.com.” This email address doesn’t actually exist, however the SMS server requires a “From” argument, so I made this one up. If a real email address is coded here, a reply to the text message will actually be sent to that email address. The subject string is defined as “Service Request.” Finally, the body is compiled of the room and item strings defined above. Here’s exactly how it’s coded:

```vba
body = """"Room: " + room + Chr(13) + "Item: " + item + """
```

As a text message, this is displayed as such:

Room: 106
Item: Air Conditioner

“Chr(13)” is just a vertical tab character in order to display these things in a visually appealing way. The only other code that’s required at this point is actually running the file with these four arguments. Early in the code, the process string is defined as the location on the computer of the SMS_Console.exe file. In order to run the file while inputting the four arguments, this line of code is run:

```vba
ID = Shell(process + " " + recipient + " " + sender + " " + subject + " " + body, vbNormalFocus)
```
Logging and Tracking Service Requests

At this point I started working on other outputs for the database. I needed to count how many times each issue came up. In order to do this, I created a “Requests” table. This table gave each request a unique ID, and kept track of the time it was created, the room number, issue, and whether or not it has been resolved.

Adding New Types of Common Issues

A few very important changes were made to the code for the form. I already allowed users to type something in the issue box that may not be in the list, but now I wanted these things to be permanently added into the Issues table. This required a few simple lines of code.

The code is run when the same button is clicked, which is still named “Command14.” The code would add the words in the issue box to the issues table if the issue not already in the table. Additionally, if the issue is already in the table, it will not add a duplicate. Here is the line of code that does this:

```
INSERT INTO Issues (Issue) VALUES ('" + item + ")
```

I also needed to add these requests into a table when the button gets clicked. Similar to the above code, I took the current items in the Room and Issue boxes and added them into the Requests table. The Requests table itself had a few more fields that would fill in automatically: “ID”, “Time”, and “Resolved”. ID attaches a unique number to each request for tracking purposes. Time records the exact time, in date format, when the request was created. The Resolved field would be a checkbox; checked means the request has been resolved, unchecked means unresolved. This by default is unchecked. Here is the code that serves this purpose:
INSERT INTO Requests (Room, Issue) VALUES ('" + room + ", " + item + "')

Call-back Timer

Finally, I added a timer to the form that would begin when the “Command14” button was clicked. The purpose of this timer is to remind the user to call the room and find out if the issue has been resolved. The duration of the timer for testing purposes is 6 seconds, but the intended duration for the final version will be 15 minutes, a number given to me by the hotel executives. At the end of the timer, a message box will come up saying “Call Room (room#): Was the issue resolved?” and will have a yes or no option. A later feature will describe the function when the user clicks yes or no.

Counting Recurring Issues

I created a query based on the Requests table, listing each issue, and the count of each issue.

Here is an example of the output from this query:

<table>
<thead>
<tr>
<th>Issue</th>
<th>CountOfId</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air Conditioner</td>
<td>5</td>
</tr>
<tr>
<td>Toilet</td>
<td>2</td>
</tr>
<tr>
<td>Television</td>
<td>1</td>
</tr>
<tr>
<td>random custom message</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 6: Requests Query

In this example, Air Conditioner has been an issue at the hotel 5 times, Toilet 2 times, etc… The purpose of this query was to be used in a Report to graphically show prominent issues in the hotel. This would help the hotel in making decisions on what needs to be improved. Here is an example of a report based on the above query:
Another function I added in this new version was a query to find all of the unresolved requests. This query was important for keeping track of the requests that still need to be resolved. The point of creating this query was to at some point use it to have a visual display on the Service Request form of the unresolved requests. An added feature of this query would be to see how much time has elapsed since the request was initiated. This is what the requests would be sorted by, giving a FIFO (First In First Out) priority system. Here’s the design view for the complete query:

<table>
<thead>
<tr>
<th>Issue</th>
<th>CountOffID</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air Conditioner</td>
<td>5</td>
<td>55.56%</td>
</tr>
<tr>
<td>random custom message</td>
<td>1</td>
<td>11.11%</td>
</tr>
<tr>
<td>Television</td>
<td>1</td>
<td>11.11%</td>
</tr>
<tr>
<td>Toilet</td>
<td>2</td>
<td>22.22%</td>
</tr>
</tbody>
</table>

Grand Total: 9

Figure 7: Report - Requests Query

Tracking Unresolved Requests
When run with sample items in the Requests table, it would output this:

<table>
<thead>
<tr>
<th>Room</th>
<th>Issue</th>
<th>Resolved</th>
<th>Time</th>
<th>TimeElapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>Air Conditione</td>
<td></td>
<td>10/30/2009 12:52:09 AM</td>
<td>616</td>
</tr>
<tr>
<td>231</td>
<td>Television</td>
<td></td>
<td>10/30/2009 12:52:26 AM</td>
<td>616</td>
</tr>
<tr>
<td>318</td>
<td>Toilet</td>
<td></td>
<td>10/30/2009 12:52:41 AM</td>
<td>616</td>
</tr>
<tr>
<td>208</td>
<td>random custom</td>
<td></td>
<td>10/30/2009 1:27:43 AM</td>
<td>615</td>
</tr>
<tr>
<td>322</td>
<td>Air Conditione</td>
<td></td>
<td>10/30/2009 1:38:44 AM</td>
<td>615</td>
</tr>
</tbody>
</table>

From left to right in the design view, I have the Room, Issue, Resolved, and Time fields being displayed from the Requests table. In order to only show unresolved requests, I inserted criteria for the resolved field “No.” This criterion allows the query to select only the requests without the “Resolved” box checked. The furthest right column was added to keep track of how much time has elapsed since the request was initiated. The column is named “TimeElapsed,” and uses the DateDiff() function within Access. It outputs in hours how much time has passed between the current time and the time of the request.
Organized Display of Time Elapsed

I needed to have a more reasonable output for the TimeElapsed column in the Unresolved Requests query. I wanted this column to keep track of units as small as seconds, but also expand to units as large as days. The obstacle was the DateDiff() function. It only outputs time as one unit: hours, days, etc. The solution to this problem was to create a new function, timediff(). This is where I consulted a very bright IE student named Tyler Peabody, who walked me through creating this function.

Here is the code I wrote for this version’s timediff() function:

```vbnet
Function timediff(starttime As Date, endtime As Date)
    Dim seconds As Single
    Dim minutes As Single
    Dim hours As Single
    Dim days As Single

    seconds = (endtime - starttime) * 24 * 3600
    minutes = Round(seconds / 60, 0) Mod 60
    hours = Round(seconds / 3600, 0) Mod 24
    days = Round(endtime - starttime, 0)
    seconds = seconds Mod 60

    If seconds > 30 Then minutes = minutes - 1
    If minutes > 30 Then hours = hours - 1
    If hours > 12 Then days = days - 1

    timediff = days & " Days, " & hours & " Hours, " & minutes & ":" & seconds
    If hours = 0 And days = 0 Then timediff = minutes & ":" & seconds
    If days = 0 And hours > 0 Then timediff = hours & " Hours, " & minutes & ":" & seconds
    If hours = 0 And days > 0 Then timediff = days & " Days, " & minutes & ":" & seconds
End Function
```

Figure 10: timediff() function code

To create the function, a new module was created. Then, the function was defined as timediff, and needed two arguments, starttime and endtime, in Date format. I define seconds, minutes, and
hours as singles for use in the function. The next few lines of code are calculations to make seconds equal to the number of seconds elapsed, minutes equal to the number of minutes elapsed, hours equal to the number of hours elapsed, and days equal to the number of days elapsed. In these calculations, I used the Round() function in Access, requiring me to write a few more lines of code to counteract when it rounds up.

Finally, I had to define what the new timediff() function would output. At first, I just had it output the number of days followed by the word Days, a comma, the number of hours followed by the word Hours, a comma, and a “minutes:seconds” format that looks like “20:15.” There were a few problems with this initial format, though. I didn’t want it to output “0 Days, 0 Hours…” So I added some code that handled when there were 0 days or 0 hours to only display items that had values.

Now, the design for the Unresolved Requests query looks like this:

```
<table>
<thead>
<tr>
<th>Field</th>
<th>ID</th>
<th>Time</th>
<th>Room</th>
<th>Issue</th>
<th>Resolved</th>
<th>Time Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID: Requests</td>
<td></td>
<td>Time:Requests</td>
<td>Room Requests</td>
<td>Issue Requests</td>
<td>Resolved Requests</td>
<td>Time Elapsed: timediff(Time, Now())</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 11: Unresolved Requests Query Design

And here is what the query outputs now, the main difference here is the “Time Elapsed” column:

```
<table>
<thead>
<tr>
<th>ID</th>
<th>Time</th>
<th>Room</th>
<th>Issue</th>
<th>Resolved</th>
<th>Time Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10/30/2009 12:52:09 AM</td>
<td>128</td>
<td>Air Conditione</td>
<td>No</td>
<td>25 Days, 17 Hours, 0:50</td>
</tr>
<tr>
<td>13</td>
<td>10/30/2009 12:52:26 AM</td>
<td>231</td>
<td>Television</td>
<td>No</td>
<td>25 Days, 17 Hours, 0:33</td>
</tr>
<tr>
<td>15</td>
<td>10/30/2009 12:52:41 AM</td>
<td>318</td>
<td>Toilet</td>
<td>No</td>
<td>25 Days, 17 Hours, 0:18</td>
</tr>
<tr>
<td>17</td>
<td>10/30/2009 12:57:43 AM</td>
<td>208</td>
<td>random custome</td>
<td>No</td>
<td>25 Days, 16 Hours, 25:16</td>
</tr>
<tr>
<td>20</td>
<td>10/30/2009 13:38:44 AM</td>
<td>322</td>
<td>Air Conditione</td>
<td>No</td>
<td>25 Days, 16 Hours, 14:15</td>
</tr>
<tr>
<td>21</td>
<td>11/3/2009 9:43:59 AM</td>
<td>106</td>
<td>Toilet</td>
<td>No</td>
<td>21 Days, 8 Hours, 9:0</td>
</tr>
</tbody>
</table>
```

Figure 12: Unresolved Requests Query Output
This is another iteration to clean up the formatting issues in this “Time Elapsed” column of the Unresolved Requests query. The Round() function in Access was not as predictable as I thought, which ended up outputting negative numbers in my timediff() function, an undesirable miscalculation. I also wanted the “minutes:seconds” format to be something people were used to seeing, “mm:ss.” Currently, it would do that when there are two digits for minutes or seconds, but if there is only one it only shows 1. As you can see in Figure 13, The last item on the list displays “9:0” meaning 9 minutes, 0 seconds. However, this is not an intuitive format to read for minutes and seconds, it needed to be displayed as “09:00.”

Here’s the improved code for the timediff() function:

```
Function timediff(starttime As Date, endtime As Date)
Dim seconds As Single
Dim minutes As Single
Dim hours As Single
Dim days As Single
Dim tminutes As Single
Dim tseconds As Single

seconds = (endtime - starttime) * 24 * 3600
minutes = Int(seconds / 60) Mod 60
hours = Int(seconds / 3600) Mod 24
days = Int(endtime - starttime)
seconds = seconds Mod 60

tminutes = Int(minutes / 10)
minutes = minutes Mod 10

tseconds = Int(seconds / 10)
seconds = seconds Mod 10

timediff = days & " Days, " & hours & " Hours, " & tminutes & minutes & ":" & tseconds & seconds
If hours = 1 Then timediff = days & " Day, " & hours & " Hour, " & tminutes & minutes & ":" & tseconds & seconds

If hours = 0 And days = 0 Then timediff = tminutes & minutes & ":" & tseconds & seconds
If days = 0 And hours > 1 Then timediff = hours & " Hours, " & tminutes & minutes & ":" & tseconds & seconds
If days = 0 And hours = 1 Then timediff = hours & " Hour, " & tminutes & minutes & ":" & tseconds & seconds
If hours = 0 And days > 0 Then timediff = days & " Days, " & tminutes & minutes & ":" & tseconds & seconds

End Function
```

Figure 14: timediff() function code
These two problems took more coding than I had anticipated. First, to solve the problem with the Round() function, I needed to switch to the Int() function, which returns the closest integer below the number given. This allowed me to remove the code that had to alter the numbers when they might have been rounded up. Now my numbers were being displayed accurately.

To solve my formatting problem with minutes and seconds, I had to define two more singles in the code, tminutes and tseconds. These new singles were the intended 10’s digit for the minutes and seconds numbers. I was then able to redefine the minutes and seconds as the 1’s digit by taking the original numbers and doing a “Mod 10” operation. Now, if you took the tminutes and minutes singles and put them together, you’d get the same numbers as before, but a 0 would appear in the 10’s digit if it would have otherwise returned a single digit (i.e. “25 = 25” still, but “9 = 09” now).

Here’s a look at the output with the above changes:

<table>
<thead>
<tr>
<th>ID</th>
<th>Time</th>
<th>Room</th>
<th>Issue</th>
<th>Resolved</th>
<th>Time Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>11/13/2009 1:07:39 PM</td>
<td>500</td>
<td>sup steve</td>
<td></td>
<td>11 Days, 5 Hours, 33:19</td>
</tr>
<tr>
<td>23</td>
<td>11/5/2009 9:39:32 PM</td>
<td>122</td>
<td>Toilet</td>
<td></td>
<td>18 Days, 21 Hours, 01:26</td>
</tr>
<tr>
<td>22</td>
<td>11/3/2009 10:26:44 AM</td>
<td>500</td>
<td>random custon</td>
<td></td>
<td>21 Days, 8 Hours, 15:14</td>
</tr>
<tr>
<td>20</td>
<td>10/30/2009 1:38:44 AM</td>
<td>322</td>
<td>Air Conditione</td>
<td></td>
<td>25 Days, 17 Hours, 02:14</td>
</tr>
<tr>
<td>17</td>
<td>10/30/2009 1:27:43 AM</td>
<td>208</td>
<td>random custon</td>
<td></td>
<td>25 Days, 17 Hours, 13:15</td>
</tr>
<tr>
<td>15</td>
<td>10/30/2009 12:52:41 AM</td>
<td>318</td>
<td>Toilet</td>
<td></td>
<td>25 Days, 17 Hours, 48:17</td>
</tr>
<tr>
<td>11</td>
<td>10/30/2009 12:52:09 AM</td>
<td>128</td>
<td>Air Conditione</td>
<td></td>
<td>25 Days, 17 Hours, 48:49</td>
</tr>
</tbody>
</table>

*Figure 15: Unresolved Requests Output*

**Unresolved Requests Subform**

After finally having an Unresolved Requests query I was satisfied with, it was time to add it into the Service Request form for a visual display. This was done using Access 2007’s wizard when I dragged the Unresolved Requests query into the design view of my MSR form. This very simple
change transformed the form from having two inputs and a button into now including a list which tracks all unresolved issues which you could check off if they do get resolved. Here’s what it looks like:

![Maintenance Service Request form](image)

Figure 16: MSR Form

I also added a new column to the Unresolved Requests query called “TotalMinutes.” This would eventually be used to track when it has been 15 minutes since the request was initiated, since the Time Elapsed was no longer a value, but a compilation of code. The plan was to use this new column in conditional formatting to turn the entire row red and bold faced. I also used this column to sort the items in the query to have the most recent request at the top. The design view of this new column looks like this in Access:

![Unresolved Requests Query Design View](image)

Figure 17: Unresolved Requests Query Design View
Conditional Formatting – Red and Bold Issues Older Than 15 Minutes

In this version, the conditional formatting of the Unresolved Requests subform was implemented. I thought I would need the “TotalMinutes” column (Figure 15, right column) to do this, because I thought conditional formatting was determined by columns. So, at first when I added this feature, I had this line of code in conditional formatting for every field on the subform:

Value.[TotalMinutes] > 15

This was the criteria to turn the font red and bold faced, and it worked fine. However I now had this unnecessary field over to the right that I found annoying. So, instead of having a column called “TotalMinutes” determined by “DateDiff(“s”,[time],Now())/60”, I took the determinant of that column and threw it into the conditional formatting criteria. Here’s what I put in the final version of this conditional formatting box:

DateDiff("s",[time],Now())/60 > 15

This allowed me to eliminate the unnecessary “TotalMinutes” column.

Call-back Reminder Added Functionality

This feature assigns a function to the pop-up box that serves as a reminder for the front desk agent to call the room back. I have a message box pop up 15 minutes after a request has been initiated that says to call the room and find out if the issue has been resolved. The message box then provides a yes or no option. Before this version, choosing yes did nothing. Now, choosing yes will check off any the request as resolved. However, the code in this version would only specify the room number in finding which request to resolve. The problem here is if a single
room has multiple requests, which is unlikely, but possible. This problem is resolved in later versions.

Here’s the code that serves this particular purpose:

```vbnet
trythis = "UPDATE Requests SET Resolved = Yes WHERE Room = " + Room + ""
Dim PauseTime, start, Finish, TotalTime
PauseTime = 6 ' Set duration.
start = Timer ' Set start time.
Do While Timer < start + PauseTime
    DoEvents ' Yield to other processes.
Loop
Finish = Timer ' Set end time.
TotalTime = Finish - start ' Calculate total time.
If (MsgBox("Call Room " + Room + ": Was the issue resolved?", 4)) = vbYes Then
    CurrentDb.Execute (trythis)
End If
```

Finally, I added some code just before this timer starts to clear the combo boxes. The purpose for this change was not for functionality, just to make the form more visually appealing.

**One Room, Multiple Requests**

This feature resolves the problem caused when one room has multiple requests. Instead of having the criteria for the update statement being based on solely the room number, I added that the issue has to be the same as well. Here’s the updated ‘trythis’ string:

```vbnet
"UPDATE Requests SET Resolved = Yes WHERE Room = " + Room + " And Issue = "' + item + ""
```

I made a simple format change for the text message body. This aesthetic change simply made sure the body was displayed below the subject. When tested on various phones, the subject would be in parenthesis before the room, but not start the body on a new line. Here’s the difference between the old display (left) and the new one (right) on my phone:

<table>
<thead>
<tr>
<th>Pre-Version 10.0</th>
<th>Version 10.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Service Request)Room: 121</td>
<td>(Service Request) - Room: 121</td>
</tr>
<tr>
<td>Item: Toilet</td>
<td>Item: Toilet</td>
</tr>
</tbody>
</table>

The code that produces this new display is the following:
I removed the footer in the MSR Form. I originally had it there to put the hotel’s logo, but there were issues with viewing the form on screens with smaller resolution because of the footer. Therefore, I just put the logo in the detail of the form and eliminated the footer.

The Form has Become the Database

A new feature was added involving when the database is opened. Now, the MSR Form is automatically displayed. This was done by going into the Access options for the current database, and selecting that it displays the MSR Form.

This is the current working version of the database. Here is a look at the final version’s form:

![MSR Form (Final Version)](image-url)
Results and Discussion

To put it plainly, the project’s results were successful. The database saves a clearly measurable amount of time for agents at the front desk. It also provides some conveniences that the original system in place lacked.

Time studies were not necessary to measure the difference or amount of time saved comparing to the old system. When the front desk agent hangs up the phone with the guest requiring service, they need only click a button. At this point in the old system, they would begin to radio the maintenance employee. The comparison in time spent from that point forward is an infinite ratio (0:00 vs. any amount of time).

Here is a summary of the advantages provided by this project:

- A well-organized list of both resolved and unresolved service requests
- Individual timers attached to each service request
- Reminders to follow up with each room after 15 minutes has passed
- Automatic communication via text message from the front desk to maintenance
- Texted data fully automated
  - Completely current – retrieves all data from in-progress form entry
  - Requires no editing whatsoever – always texts current maintenance employee
- Provides a summary of resolved service requests for future hotel improvements
The bottom line is: this database will save The Cliffs Resort a large amount of time and money. It is practically free to implement (provided they already use Microsoft Office), and is guaranteed to have a positive effect on customer satisfaction.
Conclusion and Future Improvements

Conclusion

The Cliffs Resort at Shell Beach had a problem: service requests were not getting fulfilled at an optimal rate and customer satisfaction was suffering as a result. When presented with this problem, I developed and created a working database, custom designed for the hotel. This database tracks and records service requests for the hotel. It also facilitates communication between the front desk and the maintenance staff via automatic text messaging. Implementation will result in significantly less time spent by the front desk employees initiating, tracking, and recording these service requests. It will also result in much less possibility for error when initiating these requests, increasing the rate at which these service requests are fulfilled, which will in turn increase customer satisfaction.

Future Improvements

- Currently the database has multiple requirements in order to function. First, a third party executable file (created for use with the database, SMS_Console.exe) must be present on the computer in order to successfully send the automatic text message. Second, the computer must have Access 2007 installed in order to successfully run the database. Both of these requirements could be avoided if the database were web-based. This way, people could interact with the form of my database over the internet. The server hosting this form would have SMS_Console.exe as well as Access 2007.

- The Requests Statistics Report in the database which shows the frequency of similar requests could have a better visual effect. A pie chart or bar graph would be a helpful visual aid.
• Currently the hotel only has one cell phone for their maintenance employees. If they were to expand the number of cell phones that might potentially need to receive the automatic text message, a box to select which employee to text would be an improvement for the MSR Form.

• If there are no unresolved requests listed on the MSR Form, a blank list sits on the screen, which is not visually appealing. I’d like to make the entire unresolved requests subform invisible if there is nothing on the list to improve the aesthetics of the database.
Acknowledgements

- Christopher Mathews: Computer Science Undergraduate at University of Washington

  This computer programmer was a vital source of advice when creating the executable file in C-Sharp which handles the automatic sending of text messages using an SMS Server. The file he helped code is called “SMS_Console.exe.”

- Tyler Peabody: Industrial Engineering Undergraduate at California Polytechnic State University

  This proficient Microsoft Access user helped in writing the visual basic coding required to keep track of how much time has elapsed since requests were initiated. The function he helped write is called “timediff().”
Appendix

Version 1.0

Screenshots

Figure 19 - Navigation Pane (Version 1.0)

Figure 20 – Maintenance Service Request Form Design View (Version 1.0)

Coding

(None in version 1.0)
Version 2.0

Screenshots

Figure 21 - Navigation Pane (Version 2.0)

Figure 22 - MSR Form Design View (Version 2.0)

Coding
(None in Version 2.0)
Version 3.0

Screenshots

Figure 23 - Navigation Pane (Version 3.0)

Figure 24 - MSR Form Design View (Version 3.0)
Coding

Figure 25 - VBA Command14_Click (Version 3.0)
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Mail;

namespace SMS_Console
{
    class Program
    {
        static void Main(string[] args)
        {
            if (args.Length != 4)
            {
                Console.WriteLine("Executable requires 4 arguments: To, From, Subject, Body");
                return;
            }

            MailMessage message = new MailMessage(args[1], args[0], args[2], args[3]);
            SmtpClient client = new SmtpClient("mail.jb.bugspra.com");
            client.EnableSsl = false;
            client.UseDefaultCredentials = false;
            client.DeliveryMethod = SmtpDeliveryMethod.Network;

            try
            {
                for (int i = 0; i < 1; i++)
                {
                    client.Send(message);
                    Console.WriteLine("Message Successful");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exception caught in CreateMessageWithAttachment(): " + ex.ToString());
            }
        }
    }
}
```

Figure 26 - C# SMS_Console.exe
Version 4.0

Screenshots

Figure 27 - Navigation Pane (Version 4.0)

Figure 28 - MSR Form Design View (Version 4.0)

Figure 29 - Requests Table Design View (Version 4.0)
Figure 30 - Requests Query Design View (Version 4.0)

Figure 31 - Unresolved Requests Query Design View (Version 4.0)

Figure 32 - Requests Query Report Design View (Version 4.0)
Private Sub Command14_Click()
    Dim ID As Integer
    Dim Selection As String
    Dim process As String
    Dim room As String
    Dim item As String
    Dim more As String
    Dim recipient As String
    Dim sender As String
    Dim subject As String
    Dim body As String

    Dim room = Forms!MSR!RoomBox
    Dim item = Forms!MSR!ItemBox
    Dim more = "INSERT INTO Issues (Issue) VALUES ('" + item + "');"
    Dim text = "INSERT INTO Requests (Room, Issue) VALUES ('" + room + ", " + item + ");"

    MsgBox (text)
    CurrentDb.Execute (more)
    CurrentDb.Execute (text)
    MsgBox ("Insert Submitted")
    If (MsgBox("Send a Text Message?", 4)) = vYes Then
        process = "C:\aptech\SMS_Consol.exe"
        recipient = "916348279@vttext.com"
        sender = "service@cliffsresort.com"
        subject = "Service Request"
        body = "Room: " + room + Chr(13) + "Item: " + item + ""

        'MsgBox recipient + " " + sender + " " + subject + " " + body
        'Shell(process + " " + recipient + " " + sender + " " + subject + " " + body, vbNormalFocus)
    End If
End Sub

Figure 33 - VBA Command14_Click (Version 4.0)
Version 5.0

Screenshots

Figure 34 - Navigation Pane (Version 5.0)

Figure 35 - Unresolved Requests Query Design View (Version 5.0)
Coding

```vba
Function timediff( As Date, endtime As Date)
Dim seconds As Single
Dim minutes As Single
Dim hours As Single
Dim days As Single

seconds = (endtime - startime) * 24 * 3600
minutes = Round(seconds / 60, 0) Mod 60
hours = Round(seconds / 3600, 0) Mod 24
days = Round(endtime - startime, 0)
seconds = seconds Mod 60

If seconds > 30 Then minutes = minutes - 1
If minutes > 30 Then hours = hours - 1
If hours > 12 Then days = days - 1

timediff = days & " Days, " & hours & " Hours, " & minutes & " Minutes & ":" & seconds

End Function
```

Figure 36 - timediff() Function (Version 5.0)

Version 6.0

Screenshots

Figure 37 - Navigation Pane (Version 6.0)
Figure 38 - Requery UR Macro Design View (Version 6.0)

Coding

```vbnet
Option Compare Database

Function timediff(starttime As Date, endtime As Date)
Dim seconds As Single
Dim minutes As Single
Dim hours As Single
Dim days As Single
Dim tseconds As Single
Dim tminutes As Single
Dim tseconds As Single

seconds = (endtime - starttime) * 24 * 3600
minutes = Int(seconds / 60) Mod 60
hours = Int(seconds / 3600) Mod 24
days = Int(endtime - starttime)
seconds = seconds Mod 60

tminutes = Int(minutes / 10)
minutes = minutes Mod 10

tseconds = Int(seconds / 10)
seconds = seconds Mod 10

timediff = days & " Days, " & hours & " Hours, " & tminutes & minutes & ":" & tseconds & seconds
If hours = 1 Then timediff = days & " Days, " & hours & " Hour, " & tminutes & minutes & ":" & tseconds & seconds
If hours = 0 And days = 0 Then timediff = tminutes & minutes & ":" & tseconds & seconds
If days = 0 And hours > 1 Then timediff = hours & " Hours, " & tminutes & minutes & ":" & tseconds & seconds
If days = 0 And hours = 1 Then timediff = hours & " Hour, " & tminutes & minutes & ":" & tseconds & seconds
If hours = 0 And days > 0 Then timediff = days & " Days, " & tminutes & minutes & ":" & tseconds & seconds

End Function
```

Figure 39 - timediff() Function (Version 6.0)
Version 7.0

Screenshots

Figure 40 - Navigation Pane (Version 7.0)

Figure 41 - Unresolved Requests Query Design View (Version 7.0)
Figure 42 - MSR Form Design View (Version 7.0)

Figure 43 - Unresolved Requests subform Design View (Version 7.0)
Coding
(Nothing New in Version 7.0)

Version 8.0

Screenshots

Figure 44 - MSR Form Design View (Version 8.0)

Figure 45 - Unresolved Requests Conditional Formatting (Version 8.0)
Coding

Room = Forms!NSR!Combo6
item = Forms!NSR!Combo8

more = "INSERT INTO Issues (Issue) VALUES ('" + item + "]");"
text = "INSERT INTO Requests (Room, Issue) VALUES ('" + Room + "]', " + item + "]");"
trythis = "UPDATE Requests SET Resolved = Yes WHERE Room = " + Room + "]"

MsgBox (text)
CurrentDb.Execute (more)
CurrentDb.Execute (text)

MsgBox ("Insert Submitted")
If (MsgBox("Send a Text Message?", 4)) = vbYes Then
process = "C:\apttest\SMS_Console.exe"

'recipient = "9163468279@vtext.com"
recipient = "5039156523@vtext.com"
sender = "service@cliffresort.com"
subject = "Service Request"
body = "Room: " + Room + Chr(13) + "Item: " + item + ""

'MsgBox recipient + " " + sender + " " + subject + " " + body ID = Shell(process + " " + recipient + " " + sender + " " + subject + " " + body, vbNormalFocus)
End If

Forms!NSR!Combo6.Value = 
Forms!NSR!Combo8.Value = 

Dim PauseTime, start, finish, TotalTime
PauseTime = 6 ' Set duration.
start = Timer ' Set start time.
Do While Timer < start + PauseTime
DoEvents ' Yield to other processes.
Loop
finish = Timer ' Set end time.
TotalTime = finish - start ' Calculate total time.
If (MsgBox("Call Room " + Room + " : Was the issue resolved?", 4)) = vbYes Then
CurrentDb.Execute (trythis)
End If
End

Figure 46 - VBA Command14_Click (Version 8.0)

Version 9.0

Screenshots

Coding
Version 10.0

Screenshots

Figure 47 - MSR Form Design View (Version 10.0)
Coding

```
Dim more As String
Dim currentID As Integer
Dim this As String

Dim recipient As String
Dim sender As String
Dim subject As String
Dim body As String

more = "INSERT INTO Issues (Issue) VALUES ('" + item + ":");"
text = "INSERT INTO Requests (Room, Issue) VALUES ('" + Room + ":", '" + item + ":");"
trythis = "UPDATE Requests SET Resolved = Yes WHERE Room = '" + Room + "; And Issue = '" + item + ";"

CurrentDb.Execute (more)
CurrentDb.Execute (text)

"MsgBox ("Insert Submitted")"
If MsgBox("Send a Text Message if", 9)) = vYes Then
    process = "C:\Documents and Settings\JRM\Documents\fall 09\My Dropbox\fall 09\Senior Project\EMS Console.exe"
    recipient = "8169406273@txt.att.net"
    sender = "service@cliffscemort.com"
    subject = "Service Request"
    body = "Room: " + Room + " Item: " + item + ""
    MsgBox recipient + " " + sender + " " + subject + " " + body
    Shell (process + " " + recipient + " " + sender + " " + subject + " " + body, vbNormalFocus)
End If

Forms!NSR!Combo1.Value = ""
Forms!NSR!Combo2.Value = ""

Dim PauseTime, start, Finish, TotalTime
PauseTime = 6 ' Set duration.
start = Timer ' Set start time.
Do While Timer < start + PauseTime
    DoEvents ' Yield to other processes.
Loop
Finish = Timer ' Set end time.
TotalTime = Finish - start ' Calculate total time.
If MsgBox("Call Room " + Room + ": Was the issue resolved?", 4)) = vYes Then
    CurrentDb.Execute (trythis)
End If
```

Figure 48 - VBA Command14_Click (Version 10.0)
Works Cited


Connolly, Daniel. “Lodging sustainability is about much more than just being green.” Hotel Business. ICD Publications 7 January 2009.


