2005-11-21

A Software Development Environment for Building Context-Aware Systems for Family Technology

Jeremiah Kenton Jones
Brigham Young University - Provo

A SOFTWARE DEVELOPMENT ENVIRONMENT FOR BUILDING CONTEXT-AWARE SYSTEMS FOR FAMILY TECHNOLOGY

by

Jeremiah K. Jones

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

School of Technology

Brigham Young University

December 2005

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Jeremiah K. Jones

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

_____	_____
Date	Richard Helps, Chair
_____	_____
Date	Gordon W. Romney
_____	_____
Date	Michael Bailey

BRIGHAM YOUNG UNIVERSITY

FINAL READING APPROVAL

I have read the thesis of Jeremiah K. Jones in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

_____ _____

Date

Richard Helps
Chair, Graduate Committee

Approved for the Department

_____

Thomas L. Erekson
Director, School of Technology

Accepted for the College

_____

Alan Parkinson
Dean, Ira A. Fulton College of Engineering and Technology

ABSTRACT

A SOFTWARE DEVELOPMENT ENVIRONMENT FOR BUILDING CONTEXT-AWARE SYSTEMS FOR FAMILY TECHNOLOGY

Jeremiah K. Jones

School of Technology

Master of Science

The purpose of this thesis was to utilize existing technologies to create a development environment suitable for creating context-aware applications and systems specific to home and family living conditions. After outlining the history of context-aware applications and the challenges that face family-centric systems in this field, a development environment was implemented that solves the unique challenges that face application development for family-centric, context-aware applications.

In particular, research cited in this document indicates that a browser-based user interface is the most appropriate interface for a family environment. The flexibility of the interface, as well as the familiarity of the application structure allows family members of varying levels of comprehension to use a given application. The use of a browser

interface for a context-aware system creates unique challenges such as the ability to integrate with external applications and external devices.

In addition to overcoming the restrictions of web browsers, the development environment was designed to support the unique user environment presented by a family structure. This includes mechanisms for the long-term adaptability of the system to the changing lifestyles of the family members, as well as the infrequent, but necessary ability to adjust the structure of the family unit due to the addition or prolonged absence of family members.

Another problem that the development environment was required to solve was the varying levels of computer comprehension that exist among different family members. An application that targets an entire family unit must meet the usability needs of all levels of comprehension. The development environment was created to account for this wide array of usability requirements.

The resulting development environment was implemented on a Windows XP Professional environment, utilizing existing technologies and software that were mostly cross-browser compatible. Although a new technology was not designed and created, existing technologies were combined to solve the aforementioned problems that are unique to developing systems and applications for a family-centric, context-aware environment.

Recommendations are made for future research and development in the area of family-assistive application development.

ACKNOWLEDGMENTS

I would like to thank Richard Helps for the enduring and unwearied support that he has provided to me throughout my undergraduate and graduate work, and particularly through the process of completing this thesis. His wisdom, and more importantly, his friendship have been critical elements in my completion of the masters program. I would also like to thank Gordon W. Romney and Michael Bailey for assisting me in completing this thesis and for their support throughout my coursework. I also thank the faculty and staff of the Ira A. Fulton School of Technology, and especially the faculty of the Information Technology department. You have each touched my life and education in ways that cannot be measured. You will not be forgotten. Thank you.

I also thank my friends and co-workers who have supported me in the completion of this project, and who have forgiven my flaws as a result of late nights and stressful deadlines. I especially thank Wayne Pullman for his sacrifice of time in my behalf.

Most importantly, I thank my wife, Jennifer, and my children. You mean the world to me. This degree should be awarded in your names, not mine; it is your support and love that has carried me through my education, and will continue to bring meaning to each day. I am forever indebted to you.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Background

Despite the great advances that have been made throughout many areas of technology, the concept of computer or machine learning is one that is yet to be fully implemented into computer-based systems. Although there have been several approaches developed for machine learning, they tend to be complex, expensive, or limited in scope. In general, computers are still very limited in performing tasks or functions without explicitly being "told" to do so. Although computers can perform millions of calculations per second and are capable of communicating with external devices to perform complex tasks, without proper programming or necessary input from a human user, the computer is unable to function. This is a drawback particularly when circumstances surrounding the computational device change. Computers are very limited in adapting to a changing environment.

One area where this problem stands out is that of the integration of technology in family life. Technology has become an inseparable part of daily living for most families. Technology has assisted in enhancing the lifestyles, living conditions, and enjoyment of many families. The growth of technology has provided family members with the means to efficiently complete routine tasks that might otherwise demand excessive time. Despite the many conveniences wrought by this progress, technology in the home

remains bound by the ability of the family members to operate and control the functions performed by technology. This need for further development of computer learning has especially slowed the progress of family-assistive technology.

One possible solution to the problem with computer learning is the concept of context-awareness. Recent research has focused on the idea of bringing context into computer applications in order to help alleviate some of the interfacing burden that is required by the user. If a computer can be "aware" of the context and history surrounding the application, then it can make guesses or predictions concerning the next state of the system without requiring input from a user. Context aware computing has begun to emerge specifically in the industry of mobile computing where location is used to determine the context. A simple example of this is the time synchronization that most cell phones utilize. Based on the current location of the cell phone, the phone's clock will automatically update to reflect the current time zone. However, the technology of context awareness has not become prevalent within the realm of home or family-living. In an environment such as one finds in family life, where the entire system should be built around the family members, it would be very advantageous to remove the burden of interaction from the users and embed that within the programming of the system so that it can learn and even react to the family's changing habits and living conditions. Despite the many advantages that such a system would have, a context-aware system has not yet been developed specifically for a family environment.

Based largely on the work completed by Daniel Hoopes (2004), several researchers at Brigham Young University have begun to create a context-aware home and family environment. The concept is to develop a context-aware system that can facilitate

interaction between family members and that can enhance family-focused daily living. The project, hereafter called ContexTable (Hoopes, 2004), proposes the use of external sensors and actuators to assist in determining family activities and to facilitate interaction among family members. The proposed system will utilize a large LCD computer screen to act as the system's display console. A Tablet PC is proposed to be mounted to the kitchen table in such a way as to provide user access to the system, but to not limit the normal uses of a kitchen table. The user-interface is a web-browser application in order to provide the users with a familiar environment. Trends in application development and interface design indicate that the browser-based interface is growing in popularity and commonality due to some of the standard interface elements and workflow patterns that are familiar to most modern computer users (Taylor, 2004)(Rees 2002). Especially in a family environment, where there is so much variation in computer familiarity and comprehension levels, the use of a browser interface will greatly enhance the general experience of family members of all comprehension levels.

The research team spent more than a year in an attempt to further develop ContexTable into a fully interactive family-centered system. However, from a development standpoint, there were multiple difficulties that have hindered ContexTable from making progress. The problems encountered by the ContexTable team were categorized into three groups: Issues of Context, Issues of Learning and Computation, and Issues of Technology Limitations. Each of these categories will be discussed in further detail in Chapter 3 of this document. While these problem areas will be discussed in greater detail in a later chapters, it is important to note that included within these problem areas are several crucial problems that relate specifically to developing family-

assistive technology, including: the need to provide multiple or dynamic interfaces based on the comprehension level of the family member; the need to provide a method for altering the long-term structure of the family unit; and the need for long-term learning and adjustment of the dynamic lifestyle of each family member.

The key component that has hindered the solving of these problem areas is the lack of a flexible, powerful development environment tailored to this research area. In order to overcome the difficulties presented in these three categories, especially in those specific problem areas that relate to a family structure, it is necessary to first create a development environment that takes each of these unique circumstances into consideration and allows for the development process to function smoothly while meeting the system's objectives.

Purpose of the Research

The purpose of this research is to provide a development environment that will facilitate the development of family-centered, context-aware systems that can be used within a home. The development environment will be designed in a manner that overcomes or lessens the difficulties presented by the creation of context-aware, family-assistive technology.

Research Questions

Context-aware devices are intended to provide information based on the context of a specific situation in order to facilitate greater functionality within a system, without increasing the complexity or burden of use for the user(s) of that system. The aforementioned ContexTable project is an effort to utilize the unique strengths of context-

aware systems in a family-living environment. The creation of a complete development environment that is specifically suited to these unique circumstances is essential in the effort to complete a technology-assisted, family-oriented, interactive environment. There are several questions that will be answered through this research.

First, what structure should a development environment have that meets the needs of a family-oriented, context-aware system? To create a development environment that solves the unique challenges of developing family-assistive technology that is context-aware, it is first necessary to establish the structure that is necessary to support such an environment. This research will outline the structure that is appropriate for this type of system.

Second, what existing software and technology can be combined to form a development environment to support the aforementioned structure? Once the appropriate structure for a family-assistive development environment has been decided upon, this research will then integrate existing technologies that meet the requirements of the outlined structure.

Third, what can be done to demonstrate that the created development environment successfully meets the needs for family-assistive, context-aware application development? Once the environment has been created, it will then be necessary to devise methods for testing the environment. This will demonstrate that the development environment meets the unique requirements for building family-oriented systems that have context-aware capability.

Justification

This research is important because it provides a development solution designed specifically for furthering research into the area of home and family-related technology systems. There is a current lack of technology that meets the unique needs of a family structure and family environment. This research will result in a development environment that will speed and ease the process for developing family-assistive technology. Also, it assists in further developing context-aware systems in general, particularly in the area of stationary systems (as opposed to context based primarily on location). Creating a development environment will also describe and define the problems facing researchers in this area, thus stimulating further research into possible solutions for family-assistive technology.

Methodology of the Research

The research will involve the design and creation of a development environment specifically suited to the creation of home and family-related, context-aware systems. The system will be designed using multiple existing technologies and applications in order to create a single environment that solves the unique challenges presented by context-aware, family assistive technology. In particular, the environment must be designed to accommodate a unique and changing learning algorithm as well as be able to interact with external devices and systems. The environment must also take into consideration the fact that the system must be able to initiate activities or output responses both to external applications and external devices. The environment must also allow for a web-based interface for the design of a family-friendly (as opposed to just "user-friendly") application that has two-way communication with the operating system.

The ability to adapt to the changing lifestyles and the changing structure of the family members is also a necessity for this research.

The first step in completing this process will be a review of existing technologies and applications available to meet the needs of such a system. Upon completing the review, the distinct components will then be assembled into a complete development environment. Each interface between the components will then be tested and demonstrated to exhibit the interactions that occur when the development environment is used to create a complete application. These demonstrations of components and their interfaces show the environment's ability to act as a unified development environment specifically suited to family-centric, context-aware systems.

Variables involved in the analysis will include the strengths and weaknesses of available technology, the ability of available technology to interact or integrate with other components of the system, the overall ease of development, and the relevance to the research problems that are unique to the family environment.

Delimitations

- Although simple examples for a learning algorithm were created through this research, the analysis and implementation of a learning algorithm as part of an application is not within the scope of this research.
- This research will not include the actual development and integration of the ContexTable system, or any other system. It will result only in the creation of a development environment that will facilitate the development of such a system, along with simple examples for each interface that demonstrate the usefulness of the system.

- The analysis in this research is formative in nature and does not seek to provide a complete quantitative or even a complete qualitative analysis of the resulting development environment. Simple examples of each interfacing element in the environment will be developed to substantiate the effectiveness of the created environment. The effectiveness of the system relative to the examples will be considered.
- This research will not involve the selection or integration of external sensors or applications used by ContexTable, or any similar project. Although these will be added later by other researchers, this research simulates these devices to show the development process and demonstrate the environment's ability to achieve the objectives of the research.

Assumptions

- Simple examples that demonstrate how each component in the environment solves the problems presented by the ContexTable project will be sufficient in determining the usefulness and practicality of the proposed environment.

CHAPTER 2: A REVIEW OF THE LITERATURE

Creating context-aware devices is becoming an increasingly vital element the progress of technology (Hoopes, 2004). Most technology currently requires that a system receive input from a human user before the system can take any action (Lieberman & Selker, 2000). With the advent of context-aware systems the burden of effort is largely taken up by the system (Lieberman & Selker, 2000). One area where this context-aware technology would be highly beneficial is within a family environment (Hoopes, 2004). In order to create such a system, it is first necessary to establish a stable development environment that can facilitate a team of researchers developing various components to the system, and that solves the unique challenges presented by a family structure, and the lifestyle of family members.

This review first establishes the history of context-aware computing and offers a history of past and current systems that have been created. Next, the review discusses context-aware computing as it relates to the family and home environment. The review then introduces the ContexTable project and the current status of development on that project. Next, the review addresses the problems relevant specifically to family-assistive user interfaces and development related difficulties hindering progress of family-assistive projects such as ContexTable. Lastly, the review validates the need for a development environment suited specifically to family-centric, context-aware systems.

Historical Background of Context-Aware Computing

The primary role of a context-aware system is to move most, if not all, of the system interaction from the foreground to the background (Svanaes, 2001, p. 390). Through doing this, the system becomes virtually invisible to the user. This is contrary to how computers have functioned previous to the rise of context-aware systems (Lieberman & Seiker, 2000, p. 618). By limiting the interactive effort (through time, training, attention, etc.) needed by the user, this frees the user to "turn [their] attention away from computing, per se, and toward the other activities in which computing may play a role" (Moran & Dourish, 2001, p. 92). Much of the interaction with a context-aware system should be accomplished through the normal activities of the user (Dey, 2003, p. 2).

There are various ways of classifying context-aware systems. The three that will be outlined here are: mobile, stationary, and networked.

Mobile Context-Aware Systems

Mobile context-aware systems are those that use the physical location of the device in order to evaluate the given context of the system. An example of such a system is the context-aware PDA that is used as a museum guide for the user (Petrelli, et al., 2000). Based on the location of the device, and which exhibits are nearest to it, the information given to the user will change. The interface is also customized slightly as the user interacts with it. Thus, by discovering the location of the mobile device, and by having previously received some information about the user, the system can guess the context of the situation and alter the system's output accordingly.

Another example of a mobile context-aware system is that of the Conference Assistant (Dey, et al., 1999). The conference assistant is intended to assist those attending a conference in finding and organizing a schedule of events that would interest that particular user. When entering the conference, the user fills out information about their interests and availability and the Conference Assistant is able to suggest certain events that may be of particular interest to the user of the system.

Ubiquitous computing is also being combined with context-awareness to create another realm of mobile context-aware systems. A context-aware badge that can be worn in the form of a belt is able to analyze the position of the wearer (Farringdon, et al., 1999). This could provide particular use in the case of the elderly, where if the individual is in a laying-down position at a time when they normally would not be, a system could be alerted to suggest that something may be wrong. A similar product, a woven jacket, has also been produced that can give information not only to the user's basic position, but also to more complex movements, such as throwing, hitting, and lifting (Farringdon, et al., 1999).

Stationary Context-Aware Systems

Stationary context-aware systems do not use the location of the device to determine context, rather, they use other forms of input to determine the state of the system. For example, an E-windshield will display information to users inside of the vehicle that differs from the information given to users outside of the vehicle. A user inside of the vehicle might see recreational information, geographic information, or traffic information. Externally, the windshield might be used to display information such

as advertisements or personal décor such as changeable vanity license plates (Selker, Burleson, & Arroyo, 2002).

Another example of a stationary system is that of a context-aware file system for a computer (Hess et al., 2003). This involves a file system that is able to manipulate and organize application data based on the context of the system. Such a file system is also able to import user data or values based on the context of similar files or documents. This file system also has several context-aware applications that have been ported to be able to prove the efficiency and scalability of a context-aware file structure.

A stationary context-aware room has been developed in order to further research in the area of multimodal interaction (Gieselmann, 2003). This system was developed to demonstrate how speech and gestures of an individual can be used to identify the context of the individual's situation. The research uses speech recognition and gesture recognition to discover what the individual(s) are talking about within the context of the situation. This is a first step towards the development of an "information butler" (Gieselmann, 2003).

Similar to the previous example, another stationary context-aware system has been developed for the purposes of tracking lectures and presentations in an "Intelligent Meeting Room" (Rogina, 2002). This intelligent room also seeks to act as an "information butler" by providing additional lecture or meeting information when the participants desire to obtain such. The system primarily uses complex speech recognition algorithms in order to determine the context of the lecture or meeting and summon up additional information based on the current discussion topic.

A context-aware kitchen table (ContexTable) was also built as a proof-of-concept system at Brigham Young University (Hoopes, 2004). This system was designed to simulate a kitchen experience where a family's daily routines could be learned and predicted. For example, if the user sat at the table each morning and pulled up the CNN news on their computer, after the system had determined this to be a habit, it would eventually 'learn' to pull up the CNN news based on cues from the user and the environment (such as time), rather than the user needing to perform this act individually. The table also learned other habits such as putting away groceries, paying bills, and eating breakfast. This unique, family-centered project will be discussed in further detail later in this review.

Networked Context-Aware Systems

A newer use of context-awareness has come about on the World Wide Web. As Internet use has become so popular, so has the need to customize web information according to the visitor of a site (Binemann-Zdanowicz et al., 2004). An example of such a system is a home-loan application system. Such a system is complicated in that "the applicant [is] not necessarily exactly one natural person" (Zdanowicz et al., 2004). Also, each applicant's specific debts and properties need to be taken into consideration. In this system, the information displayed to the user is based upon the user's own situation and not based solely on a simplistic rule set. Previous to such a context-aware application, most home-loan applications needed to be processed in person to deal with the complexity of each application.

Another recent networked system is the Portable Help Desk (Anhalt et al., 2001). This is a system that was developed at Carnegie Mellon University that involves the use

of multiple context-aware devices that communicate via a non-standard protocol over standard wireless access points. The system is able to allow users to locate and easily communicate with associates. The system is able to determine the location of individuals and helps to coordinate meeting places when individuals desire to group together. It is also able to provide limited information such as telephone, address, and other contact information for the users.

Family-Centric, Context-Aware Systems

It is hardly disputable that information technology has become an integral part of family living (Hughes, 1999) (Statistical Research, Inc., 1998). Technology in the form of telephones, cell phones, mobile devices, personal PCs, appliances, entertainment, and a variety of other forms has penetrated the majority of American households (Hughes, 1999). Recent research has indicated that the onslaught of many forms of technology, particularly relating to the media (Internet, television, radio, etc.) has had a negative impact on family communication and relationships (Kraut et. al, 1998). This research indicates that the use of the Internet has particularly led to decreased familial communication and an enhanced feeling of disparity and loneliness among the family members. Furthermore, it has been established that healthy interaction and communication between family members is integral in the normal development of children in a home (Dunst, 2002).

More recent research has been performed on the affects of technology on family communication (Chesley, 2005)(Jackson, 2005). The cited research indicates that while technology makes both work and family communication more accessible via cell phones, E-mail, and pagers, this same ease in communication leads to lessened trust in a

relationship (Jackson, 2005) and contributes to increased dissatisfaction with family relationships, especially among women (Chesley, 2005).

Although this thesis is not focused on the effects of technology on family interaction, the previously cited research raises a concern that technology is not being utilized specifically to strengthen family communication and family relationships, but rather, it is often inadvertently weakening these relationships. As has been stated previously in this review, one of the major benefits of context-aware systems is that they are intended to enhance communication and interaction between individuals. Context-aware computing can be integrated with family assistive technology to begin developing systems and applications intended to strengthen family relationships and assist in healthy family communication.

There is currently extensive research being performed around the idea of building a "context-aware home" (Rosen, 2004) (Myer, 2003). Current research in this area is geared towards adding context to homes to ease daily living of individuals. Much of this research is devoted specifically to the idea of adding context to the multi-media environments found in homes. Despite the focus on adding context to home-life, this research lacks a focus of building context-aware applications for the enhancement of family communication and strengthening of family relationships. While the concepts of "home" and "family" are often used simultaneously, it is critical to this thesis to establish the fundamental differences between the concept of a "home" and that of a "family." A home is a structure where one or more individuals may reside. A family is a unit of society that is based on unique relationships. While the term "family" can be used quite loosely, for the purposes of this thesis, a family is defined to be a legally married man

and woman with zero or more children. Current research into the area of context-aware homes does not focus on family relationships, but rather focuses on the concept of a home as a living structure. In "A Survey of Research on Context-Aware Homes," (Meyer, 2003) Meyer and Rakotonirainy discuss the fact that context-aware systems were built primarily for the work environment. This research then goes on to explore the concept of building homes that are context-aware. This article is quite typical of the type of research currently being performed in this area. The following excerpt from this paper is an example of the type of research currently being explored:

"The goal of research on context-aware buildings is to offer an unobtrusive and appealing environment embedded with pervasive devices that help its occupants to achieve their tasks at hand; technology that interacts closely with its occupants in the most natural ways to the point where such interaction becomes implicit."

This excerpt, which is typical of current research, emphasizes the home as a structure in which individuals or "occupants" exist. There is a current lack of specific research into the area of integrating context-aware technology into the family setting as it has been previously defined. Therefore, although context-aware systems for home technology are under development, as of the date of this research, no record of a context-aware system has been found for use specifically in strengthening family relationships and enhancing family communication.

Introduction to the ContexTable Project

Although aforementioned research indicates that some forms of technology have a negative impact on family living, the ContexTable project (Hoopes, 2004) is making an

effort to expand the uses of context-aware computing into the home and family to help enhance a family's ability to interact and communicate with one another.

The ContexTable project was initiated by Daniel Hoopes as a thesis project at Brigham Young University. The project began as an effort to experiment with context-aware computing as a stationary, home-based system. Further research is heading to the development of a context-aware system that will facilitate communication and interaction between family members. Currently, the ContexTable project is experiencing slow progress due to several problems relating to development procedures. These problems and hindrances will be discussed in greater detail later in this review.

Establishing the Need for a Home-Centric Development Environment

Increasing Complexity of Context-Aware Systems and the Lack of Established Development Environments

As context-aware systems become more prevalent in daily life, the need for these devices to effectively communicate with external systems and devices becomes correspondingly more important (Capra et al., 2001). For example, to add context-awareness to a home one might want to have an entertainment system, lighting, and communications services that all interact with family members (users) and with each other. These seemingly separate devices might benefit from having a centralized, context-aware system that manages interaction between the devices, and the family members. For example, if the entertainment system is triggered to begin playing the television while the kitchen table is triggered to begin pulling up the internet-news, it would be beneficial for these devices to communicate with one another to determine if there might be a conflict in the triggered events. A shared communication may be able to

determine that the events were indeed triggered by the same family member and that the user may need to give feedback to indicate which (if any, or both) of the events should be executed.

There has been limited research in the area of standardized environments for context-aware systems. One such research project involved the creation of CAMP, which enables "interactions among users and service providers through the mediation of the environment" (Mandato, 2002, p. 92). This system is proposed to provide mobility of context-aware devices within a limited range. It uses a network backbone, adaptation engine, and a service discovery mechanism in order to aid in the communication of mobile users. One postulated result of such communication could be used in the world of multimedia to provide a user with local movie times and information based on the previous context of the user. This involves limited communication between a movie theater's context system and that of the user's mobile context system.

Another example of previous research in this area is that of CARISMA: a Context-Aware Reflective mIddleware System for Mobile Applications (Capra, et al., 2003). This research set out to define a primitive set of middleware for software engineers to describe how to handle context changes based on predefined policies. This research focuses primarily on the use of position-aware systems and seeks to find a way to describe locale changes above those of other context alterations. This work is particularly valuable in determining and resolving conflicts between communications. CARISMA argues that communication conflicts between context-aware devices should be handled at execution time and not handled statically at design time. To resolve conflicts, the middleware treats the system like a micro-economy where services and

devices make “bids” on the quality of service that they demand. The middleware then “auctions” off the service to the highest bidder, thus resolving conflicts by priority of service. Although this means for conflict resolution may not be the ideal solution, this literature demonstrates the interest for research in the area of networking multiple context-aware devices together.

MobiPADS is another example of an attempt to provide middleware for context-aware mobile environments (Chan, & Chuang, 2003). This system is derived from an event-notification model where the middleware adapts to changing context based on event flags. It is based on standard XML document formats that provide context information to the system. Once again, this research is focused on mobile context-aware devices and is limited in providing information between other forms of context-aware systems.

Although forms of communication have been discussed concerning web service (or networked) context-awareness (Maamar et al., 2004) and researched in limited form, widespread standardization for development environments suited to either mobile or stationary context-aware systems has not been established. Furthermore, no evidence of research specific to building context-aware, family-oriented systems has been found. None of the systems described previously provide detailed information regarding the actual development environment for the projects. The difficulties involved in creating such a system have not been confronted directly. Therefore, this document serves to initiate future literature regarding the topic of family-centric, context-aware systems, and the development environment necessary in order to create and deploy such a system.

Family-Friendly User Interfaces

For the original ContexTable project (Hoopes, 2004), the user interface selected for the project was not an integral part of the research. As the ContexTable project grew, it was realized that the user interface would greatly affect the ability of the family to interact with the system. It was quickly realized that to develop systems specific to the family setting, the type of user interface selected becomes critical to the family's overall experience with the system. A family is made up of various demographics of users; there are differences in age, sex, and comprehension levels among each of the family members. For this reason, it is important for the family-assistive projects to have an interface that meets the various needs of the family members.

According to recent research, it appears that the Internet Browser interface is becoming one of the most widely used interfaces (Taylor, 2004). Not only do web application developers utilize a browser interface, but many software developers are beginning to do the same (Taylor, 2004). Due to an increase in the ease of developing on the browser platform, the ability for web applications (even when running only on the desktop) to perform complex tasks is increasing dramatically (Rees, 2002). To quote Michael J. Rees:

"If current trends continue, it is likely that the web browser will become the only widely used user interface."

One of the primary benefits of the browser interface is that the basic concept of the browser has become quite standard to most computer users. The browser interface provides a familiar and simple way of communicating an application's processes to users of multiple demographics. The familiarity of the browser assists in creating a system that

is largely invisible to the family members. By combining a context-aware environment with an interface that is familiar, the burden of technology is eased. The browser interface also allows for the development of extremely dynamic and flexible interfaces using standard technologies that are widely accepted such as HTML, XML, Cascading Style Sheets (CSS), Macromedia Flash (or other interactive technologies), XSL, and others. Through the browser, these technologies can be mixed and matched to provide each family member with an interface that is suited to their preferences and comprehension level. For this reason, it is critical that this research result in a development environment that utilizes a browser-based interface.

Problems Hindering Family-Assistive Technology Development

As previously mentioned, the ContexTable project is currently suffering from several development problems that must be overcome in order to successfully create a family-centric, context-aware system. The problems encountered by this project have been categorized into three groups: Issues of Context, Issues of Learning and Computation, and Issues of Technology Limitations. This research will supply a development environment that resolves or accounts for all of the problems encountered within these three categories. The problems hindering context-aware, family-centric systems will be discussed in greater detail in Chapter 3 of this document.

CHAPTER 3: METHODOLOGY OF RESEARCH

Methodology Overview

The desired outcome of this project is the creation of a development environment that will facilitate the unique challenges involved in building a family-centric, context-aware system. To accomplish this task, the research methodology is split up into two categories: Defining the Challenges and Solution Demonstrations.

Defining the Challenges

As mentioned briefly in Chapter 2, there are several unique problems that creating a home-centric, context-aware system presents. These problems have been categorized into three general areas and will be clearly defined throughout the remainder of this section. The problem areas are: Issues of Context, Issues of Learning and Computation, and Issues of Technology Limitations. Figure 3.1 represents the very basic structure of a context-aware system designed specifically for a family environment. This system includes several important components that will be described in further detail (in no particular order) for the purposes of elaborating on the challenges involved in creating such technology.

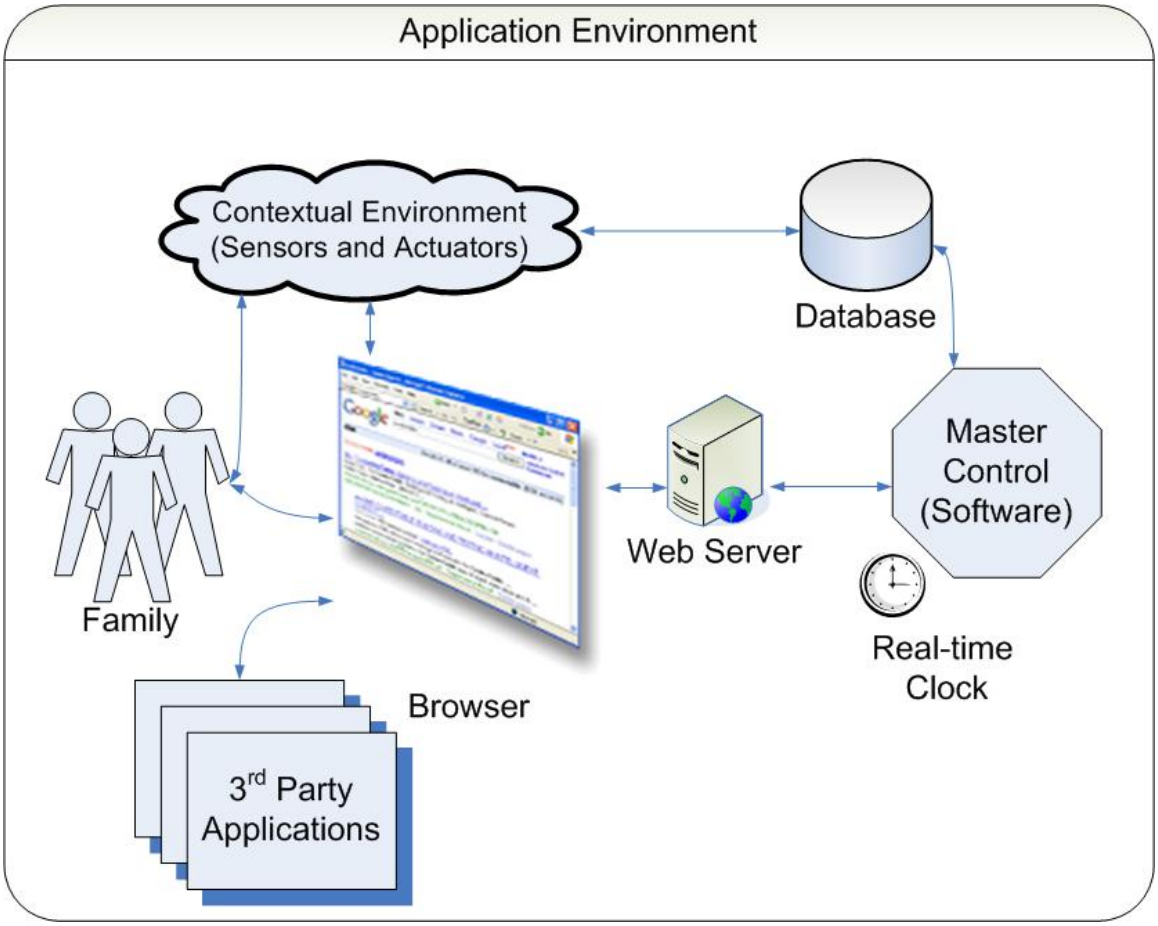


Figure 3.1 - Basic Structure for Context-Aware, Family-Centric System

The first component in this environment is the ubiquitous environment which represents the external hardware, sensors, and actuators that assist the system in determining context. This environment is the primary medium the system uses to determine context of the current situation. While this is not the only area where context may be analyzed, it represents the major portion of the system where contextual information will be discovered. For this reason, this ubiquitous environment is hereafter referred to as the Contextual Environment. This portion of the technology structure should be ubiquitous, in that the user is hardly aware of its existence. The development

environment does not include the creation or implementation of the Contextual Environment, due to the extreme variation that one system may implement from that of another. However, the development environment must allow for the existence of this environment and provide a methodology for its implementation in a family-centric system.

The second component is the family. These are the people (users) that will interact with the system, and for whom the system is being designed. The family unit presents a unique set of challenges to the context-aware environment. First, due to the wide variations in user expertise and preference, the interface for the system must adapt to the preferences and needs of each family member, on an individual basis. Second, as the family structure changes, the system must appropriately handle the addition or removal of active family members without requiring major application changes or development alterations. Third, the system must have a long-term method for adapting to the changing lifestyle, habits, and comprehension of family members.

The browser interface represents the third component in the environment. This interface provides the flexibility and familiarity required in creating a system that can be used by the people with widely varying demographic profiles found within a single family unit. This interface will be combined with the previously described Contextual Environment to gather information about the family and the context of the current situation.

To support the browser interface, the fourth component required is a web server. There are many web servers available that can be installed and used on a regular desktop PC. The web server must provide normal server functions and also capture user events

and then interact with the operating system. In order to simplify the system, the Contextual Environment (sensors and actuators) may also interact directly with the web server via standard HTTP GET requests, thus simulating interaction as if it were coming through a browser. The environment should also allow the Contextual Environment to directly access the database, to allow future researchers flexibility in developing this portion of the system. The environment must involve the creation of an API that the Contextual Environment may utilize to interact with the system. Using this simplified interface, the only requirement of the Contextual Environment is that it has either the capability of communicating using the HTTP protocol, or that it has the ability to interact with the database.

The fifth component is the "master control." This software component represents the learning mechanism for the system. The development environment must provide modularity such that the learning algorithm can be easily modified or replaced without needing to reprogram or redesign the system. It is anticipated that future application may require real-time capabilities for some activities. In addition, a time-of-day clock may be necessary to track context and to schedule events. Although the time-of-day clock will be addressed for scheduling purposes, the real-time response of the system will be left to the normal prioritization schedule and processing directives of the host operating system. As this is not designed to create life-critical systems, the real-time directives built into the host operating system will be adequate for creating an acceptable experience for the family.

The master control will be attached to a database (the sixth component) which will house all of the data necessary for the system to run. The data is stored as a separate

component in order to provide future growth, development, and analysis of the best software for actually storing the data. This research does not seek to define the best software for storing data, but does require that an appropriate data storage schema and methodology be defined. Future researchers may select alternate databases, so long as it supports relational database interaction, which is required for the proposed development environment.

The seventh component is represented by the 3rd party applications that must also interact with the system. In addition to being able to interact with the family and the Contextual Enviroinment, the system must also interact with 3rd party applications or software on the operating system. Due to the large number of software packages already available for supporting the lives of family members, it is not intended for the system to involve the creation of such software systems. Therefore, the development environment must allow for limited two-way interaction with 3rd party applications. Applications are produced by many developers with no standards for external interaction. The complexity involved with creating a communication standard between applications is such that it is not reasonable to pursue completely general two-way interaction within the scope of this research. For that purpose, the current system only provides for *limited* two-way communication between the system and 3rd party applications. Limited in this context means, the environment must be able to execute and terminate external processes. No provision is made for passing data into or out of 3rd-party applications.

Having defined each of the components in the system, the following sections will further discuss the three problem categories, and which category each of these components falls into.

Issues of Context

This development environment must be designed specifically for use with context-aware environments. This topic has already been discussed in great detail. However, it is important to note that the proposed development environment must take into consideration the uniqueness of a context-aware system. This includes being able to interact with external hardware, sensors, and actuators, as well as provide a learning mechanism for recognizing behavior (context) and making decisions based on the context. Although the actual design and implementation of this "external" environment should not be determined by the development environment, the proposed development environment must appropriately handle interaction with these sensors and actuators. This will involve the creation of a standard API or standard communication methodology in order to appropriately handle external devices.

Along with accounting for the Contextual Environment, the development environment must also track context based on the family's direct interaction with the web browser that interfaces to the system. As the user interacts, the system must derive context from the family's current and past actions. Thus the development environment must facilitate the direct interaction with family members as well as recording contextual information.

Issues of Learning and Computation

One of the most dynamic problems encountered by the creation of a family-assistive, context-aware technology relates to the learning mechanisms that must be developed for use in the system. The system must learn to interact with the family members on an individual level as well as a familial level. The habits and tendencies of

both individual family members and the family as a unit must be recognized and recorded by the system. These recorded tendencies must then become interactive to the point that the system can assist the family in daily living and communication. The system must also have the ability to adapt on a long-term basis to the changing lifestyles of each family member. The learning algorithm must also account for the fact that the family structure may change from time to time. As children grow and move out of the household, or as new family members are brought into the home, the system must adapt to the changing family structure without requiring development system changes.

As the learning algorithm becomes better defined, it is expected that the algorithm may change frequently (particularly in the early stages of development). It is also likely that multiple algorithms will be tested with the system in a process to determine the best algorithm to utilize. For this purpose, the proposed environment must be flexible and modular so that the learning algorithm can be altered and adjusted without major remodeling of the entire system's infrastructure. This will require that the development environment provide a plug-and-play type of interface for the learning mechanism.

The development environment must also define a schema or method for storing data. As the system tracks the family's actions and activities, a vast amount of data will be collected and must be stored in a way that keeps the data storage technology separate from the remaining system. This must be done in order to allow for upgrades or alterations in the software chosen for data storage. Along with this, there must be consideration for the fact that the data stored must occasionally be "cleaned up" or discarded. As data is collected and stored, not all of it will remain relevant to the family. Therefore, the development environment must allow for pre-defined, but adjustable time

intervals at which unnecessary records will be discarded. This component should be as modular as possible, in order to allow for future refinement and development on how the data is handled. It is not within the scope of this research to solidly define the logic for how the data should be treated so it will just be stored for one month for the purposes of demonstrating the environment's ability to handle storage of data for a variable length of time.

Issues of Technology Limitations

Although there are many context-aware systems that are confronted by technology limitations, there is one particular area that presents a considerable dilemma when attempting to create a family oriented environment—web browser limitations. As was discussed in Chapter 2, the web browser is an excellent user interface for a family system. The environment is familiar, adaptable, and appeals to users of various levels. Although there are many options for developing a web-based application, the primary problem that this project presents is how to handle interactions between the web browser and the operating system. As internet security awareness has increased, so have the security restrictions placed upon web browsers. Many browsers currently limit interaction with the host operating system by preventing execution and termination of external processes and applications.

Family-assistive systems such as will be created through this development environment require that the user interface be able to cause events external to the browser to take place. The project also requires that the operating system be able to somehow monitor, or "be aware" of what is happening in the browser. The development environment must compensate for the violation of the browser security sandbox by

providing a means for the web browser to have limited, two-way communication with the operating system. The details of what is meant by "limited, two-way communication" will be discussed further in a later section of this document.

Furthermore, the system must have the ability to execute and terminate 3rd party applications. The overall project does not seek to write applications to enhance the daily living of family members, but rather, it seeks to utilize existing applications and combine those applications with a context-aware environment such that the family can easily interact with already existing systems. The development environment must provide a way for the system to execute and terminate applications relevant to the family members.

Solution Demonstrations

Having determined the various components and interfaces in the development environment, the final task is to implement this environment by testing the ability of each component and solution to interact with and interface with the environment. To prove the successful implementation of the development environment, it is not only necessary to demonstrate that each component of the environment is functional or accounted for, but also that each interface between components can be demonstrated. By demonstrating successful communication across these interfaces, it can be concluded that the environment can be utilized to create a complete system based upon these basic interactions. More complex interactions may require specialized modification to the environment.

Furthermore, each of the three primary problem areas involved in family-centric, context-aware computing (as described previously) must be solved and/or accounted for. To accomplish this, a variety of demonstrations will be created that highlight each of the

problems presented through this research and how the development environment has solved or accounted for that problem. Through successful completion and evaluation of these demonstrations, it will be determined that the development environment has adequately eliminated or overcome each of the unique problems presented by a home and family-centered, context-aware system. The remainder of this chapter outlines each proposed demonstration.

General Development Considerations

The proposed development environment will be built on a Windows XP Professional platform and will utilize a MySQL database, Apache web server, and PHP scripting language. The proposed system will be demonstrated only in Internet Explorer although the system will be flexible enough to run via most of the popular browsers (i.e. Netscape, Safari, Firefox, etc), or should provide mechanisms for cross-browser compatibility by future researchers. All versioning and collaboration will be handled through a Concurrent Versions System (CVS) server. The demonstration platform will run on Microsoft Virtual PC as a virtual operating system.

The Apache/MySQL/PHP combination was selected due to the flexibility of this system to be deployed on multiple platforms including Windows, Linux, Unix, Sun, PowerPC, and others. Although the scope of this research will not explore cross-platform compatibility, it was decided to design the system in a way that would provide future researchers with the possibility of porting the application to an alternate system other than the Windows XP Professional system selected for the purposes of this research. The Apache/MySQL/PHP combination also yields several advantages due to the open source nature of the software. All three of these products are open-source and thus provide a

vast source of informal documentation and code modules that can be implemented into the environment by future researchers.

CVS was selected for versioning as a result of the widespread use of CVS on large-scale development projects including such projects as MySQL, Apache, and many other widely used products. CVS provides a way for multiple team members to "check out" a copy of a code repository to make local changes without needing to interfere with another developer's work. CVS also uses differential software to assist developers in resolving conflicts between versions of source code.

The demonstration system was deployed on a virtual server to allow for easy portability of the entire system from one computer or location to another. By building the system on a virtual hard disk, and virtual operating system, it can be saved on an external device and moved from one computer to another without needing to re-configure the system. This allows for a more simplified demonstration process. The development environment does not require implementation on a virtual server, but was implemented in this manner only for the purpose of easy portability and reproduction for research purposes. It should be noted that while development is not hindered by the use of a virtual server, a completed application could notice a significant degradation of performance if implemented on a virtual server; for this reason, it is recommended that future developers not implement completed applications on a virtual server.

It is also important to mention security as a part of the overall system. As a development environment, the security risks do not lie as much in the environment as in the application that is deployed. For this purpose, almost all security considerations must be handled by future developers utilizing the environment to create an application. Thus

it should be noted that future developers and researchers should be aware of any security risks inherent in the application being developed, and take appropriate measures to ensure that the application is adequately secure.

Browser Launching an External Application

As explained in previous sections, the use of a browser as an application interface limits the application's ability to interact with the native operating system and external applications. In order to overcome this limitation, the development environment will utilize a 3rd party ActiveX control called "LaunchinIE" (Mannaerts, 2003). This control allows specified URLs limited communication with the native operating system. By specifying the "http://localhost/" URL as the only acceptable controller, the system remains Internet-safe while allowing the application to communicate with the browser. In order to demonstrate this, a simple example will be created that shows the development environment's ability to utilize this ActiveX control to execute external applications as well as demonstrate how the system remains protected from Internet-based systems accessing the functionality provided by this ActiveX control.

Browser Terminating an External Application

Similar to allowing the browser to execute an application, family-centric, context-aware systems require the ability to terminate an application. While the previously described ActiveX control will allow for the execution of an application, it does not inherently terminate an application. To control the termination of an application, PHP functions will be written to control native process listing and process termination. These functions will then be incorporated into a simple web demonstration that shows the

browser's ability to call these functions, subsequently calling native operating system functions that can then terminate a process on the local machine.

Modularity of Learning Algorithm

In order to demonstrate the modularity of the learning algorithm, two very simple, but different learning algorithms will be written. The demonstration will then show how the learning algorithm can be quickly removed and replaced by another algorithm without modifying other components or modules in the system. The original module will be replaced by the second module without restarting or reconfiguring the system and a simple demonstration of the input and output of the algorithm will adequately illustrate the development environment's ability to account for a changing learning module.

Behavior Recognition

Although behavior recognition is part of the learning algorithm, it is a critical portion of the system that is distinct and complex enough that it deserves special mention. The development environment must allow for the learning module's behavior or pattern recognition system to be modified and replaced without disrupting other parts of the system. Because this is so closely tied to the learning algorithm, a separate demonstration will not be implemented for the behavior recognition. The manner in which the behavior recognition module interacts with the rest of the system is identical to the manner in which the learning algorithm interfaces with the rest of a completed system. Thus it is not necessary to demonstrate this interaction. Based upon the results of the learning algorithm demonstration, the structure of the learning module and behavior recognition module will be described in a way that illustrates the modularity of

both modules. Because the interface between the learning module and the rest of the system is the same concept and practice as the interface between the behavior recognition and the system, successfully demonstrating one will also prove the success of the other.

Ability to Receive Input from an External System

The system must have two-way communication with external systems such as the sensors, actuators, and other devices that might be used in the development of the system. The development environment will allow external systems to send input to the application through two methods: direct database access and a web service. The database structure will be set up in a way that an external system can directly access specific tables in the database that indicate to the application the current status of external devices. If direct database access is not desired, or not available, the development environment will also include a custom web-service that will allow any external system to make HTTP GET or POST requests that will then perform specific actions such as updating the system status, or inputting information into the system. To demonstrate these abilities, PHPMyAdmin, a web-based database utility, will be used to show how an external system can directly access the database and change the state of information therein (so long as the external application has the appropriate rights to do so). In addition to this, a demonstration of the custom web-service's ability to communicate with the system will be created that shows how HTTP GET requests can be sent to the service to indirectly modify information in the database.

Responding to Events and Sequences of Events

The development environment must be flexible enough to allow for unforeseen modifications to what might comprise an event or behavior. To accommodate this, the development environment will define a "Pattern" or behavior as a combination of Actors, Events, Devices, and other unforeseen variables. The development environment will be built to allow the system to automatically adapt to new variables that may become relevant, such as season, temperature, etc. Although the demonstration will not include these extraneous variables, the development environment will be built in a way that accommodates changing behavior criteria. If the criteria for defining a pattern are altered then only the learning algorithm and behavior recognition modules will require modification. Having designed the system in this manner, a sequence of events will be handled the same as any other pattern, but will be comprised of previous events as if it were one single event. Only the behavior recognition module will need to be modified to handle the parsing and recognition of event sequences. The scope of this research does not seek to demonstrate event sequencing recognition, but only seeks to demonstrate how events are handled by the system, and how a sequence of events can be considered a single event of a special type. To demonstrate the ability for the system to handle events, PHPMyAdmin will be used to show the state of the database both before and after the demonstration. The demonstration will be comprised of clicking on a web page that has a variety of options for the user to select. When the user clicks on a specific option that is considered an event then the system will use the learning module to learn and recognize the user's behavior. This will be demonstrated by showing how the database was modified according to the learning algorithm's specifications.

In addition to a user directly triggering events in the system, the external devices must also be able to trigger events. To demonstrate the development environment's ability to allow external systems to trigger events, a simple demonstration of how the web service may be called via HTTP GET or POST will illustrate the system's ability to trigger events without needing the user to directly click on a link to activate an event.

Similar to allowing an external system to trigger an event, the development environment and web service will also include the ability to scan the current environment and status in order to look for a behavior or pattern match. This process can then be triggered by a scheduled task or externally scheduled application to allow the system to be constantly checking and updating its status. The scope of this research does not include the creation of a scheduled task or scheduled application.

Code Modularity and Configurability

A successful and well-designed development environment will be built in a way that makes the system easy to configure and modify. In order to accomplish this, the development environment will be primarily Object Oriented and built using modules and functions as opposed to less-structured scripting. The environment should also contain some sort of global configuration file that allows developers to quickly reconfigure the system to perform in different ways. It should be built such that unforeseen configurations can be easily added by future researchers and developers. To demonstrate this, an overview of the application architecture will be provided that shows how the application has been built in a way to meet the needs of a modular and configurable application.

Dynamic Family Structure and Lifestyles

The system must accommodate for multiple family members, as well as a changing number of family members. The development environment will be built in a way that allows for family members (and other users) to be added to or removed from the system without needing to modify source code. A demonstration will be executed that shows how adding a new user or family member to the database will automatically integrate that new user into the system without needing to modify the source code. By doing this, a future researcher or developer can create a custom interface that allows the family to easily modify its household members and guests for the system.

The environment must also accommodate the long-term lifestyles of each family member. As time goes on, the lifestyle, habits, preferences, and comprehension levels of the family members are likely to change. Once the system has learned a certain behavior, the environment must allow for the flexibility to change that learned behavior in accordance with the family member's evolving lifestyle. To demonstrate this, a description will be provided of how the development environment combines each of the components necessary to provide this functionality. A hypothetical scenario that includes the details of how the development environment adapts to the new lifestyle will also be provided. This will conceptually demonstrate how the componenets of the system interface to allow for adaptation to lifestyles and learned behaviors.

Family-Friendly Interface

A typical family can be comprised of family members of a wide variety of ages ranging from infant to mature adult. Within those demographics, there will be various levels of comprehension and wide differences in preferences. This creates a unique

challenge for building a family-centric, context-aware system. To accommodate this need, the development environment must support multiple, dynamic interfaces that can change based on a family member's specific needs and preferences. This requires the system to learn a family member's behavior, or to at least provide set templates that may be selected for that family member. Although the scope of this research does not include developing or deploying a learning system, nor does it include the creation of dynamic or varying interfaces, the development environment must be architected in a way that will allow for this type of research and development in the future. The development environment must have the capacity to allow future developers to create family-friendly interfaces that can vary from user-to-user. By keeping a standard MVC (Model-View-Controller) programming methodology, the development environment can be built in a way that allows for multiple interfaces to be attached to the system without modifying the core functions and modules of the system. A definition and description of the MVC architecture will be supplied in greater detail in Chapter 4.

To demonstrate this architecture, two basic interfaces will be built that are very different from one another. These interfaces will be simple, yet different enough to demonstrate the application's ability to adapt to virtually any interface. The interfaces will include a dynamic list of applications that the user can select to have the system execute. By clicking on an application, the system will "learn" the behavior and then execute the application. This will be proven by examining the database both before and after user interaction. The demonstration will be repeated for both interfaces, thus proving the system's ability to provide multiple, dynamic interfaces in support of the varying needs of family members.

CHAPTER 4: RESULTS AND ANALYSIS

Review of Success Parameters

In order to determine that the results of this research were successful, it is important to demonstrate that each of the components in the development environment have the ability to function within a system. To demonstrate this, the successful implementation of each component as well as its interfaces is necessary. Because the demonstration of each component alone does not demonstrate the successful integration of the complete system, it is also necessary to provide demonstrations that show how each component interfaces and interacts with the rest of the system. If the environment is proven to effectively communicate through each interface between components, then it can be utilized as a single environment for developing complete applications. Although the design and architecture of a completed system was not within the scope of this research, the success of this research will greatly aid in the development of family-centric, context-aware systems by future researchers. By providing a stable and scalable development environment, future researchers may fully implement the ContexTable project as well as other family-assistive technologies. The parameters for determining the success of the system was to demonstrate the following:

- Code Modularity and Configurability
- Dynamic Family Structure and Lifestyles

- Browser Launching an External Application
- Browser Terminating an External Application
- Modularity of Learning Algorithm
- Behavior Recognition
- Responding to Events and Sequences of Events
- Ability to Receive Input from External System
- Family-Friendly Interface.

The successful demonstration of each parameter as it interfaces with its immediate components is assumed to prove the potential of the whole system as a development environment that meets the requirements for the creation of family-centric, context-aware systems. A summary of the results of interface demonstrations is provided at the end of this chapter.

Demonstration Results

Infrastructure: Code Modularity and Configurability

One of the most important aspects of a scalable development environment is to architect an infrastructure that is modular. It should also have a mechanism for organizing global configurations that can affect the entire system without needing to modify individual modules. This development environment was architected in a manner that not only allows for, but essentially requires modularity. The infrastructure was designed in a way that allows for extreme flexibility so that multiple types of family-

centric, context-aware applications may be built around the core infrastructure. The architecture of the application is outlined and described below.

PHP Hypertext Preprocessor (PHP) was selected as the programming language of choice due to its flexibility in web application development and its widespread usage and availability. Figure 4.1 graphs the usage and acceptance for PHP as a web programming language over the span of the last five years. This graph shows the steady increase that PHP has maintained as an accepted web programming language.

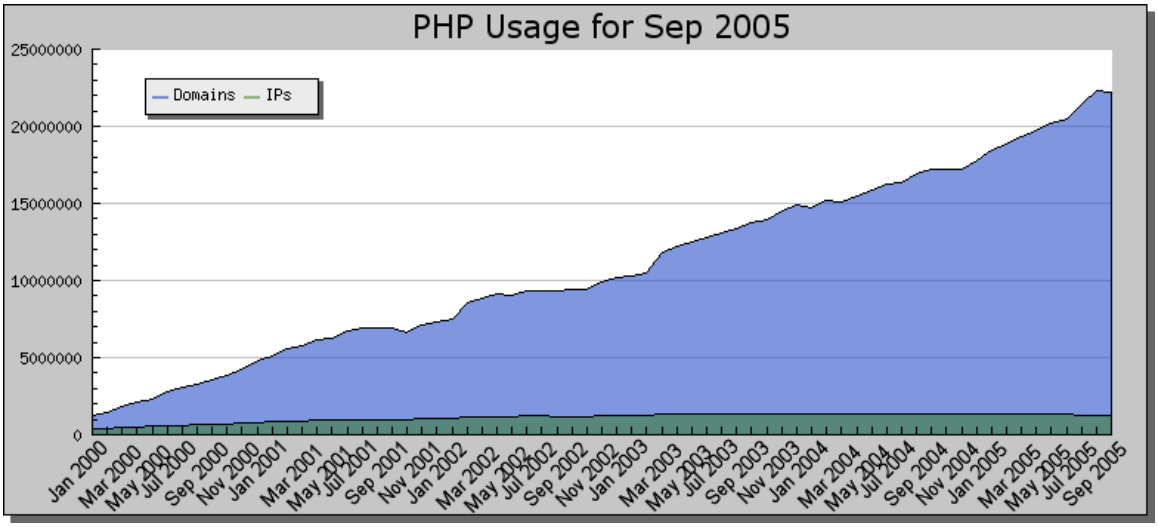


Figure 4.1 - PHP Growth Statistics (PHP, 2005)

By selecting a programming language that is widely used and widely supported, the projects created by the development environment may be more easily ported to alternate platforms. The wide usage of PHP also allows developers to tap into the vast source code repositories and modules that are made available for public use through the Internet. PHP also has advantages over other web languages such as Microsoft C#.NET

or Java due to the fact that it does not need to be recompiled after each modification. This can significantly decrease development time required for creating an application.

The application was architected following a typical Model-View-Controller (MVC) infrastructure (Hansen, 2005). While this research is not an in-depth look at MVC architecture, a brief explanation of this method for programming as applied to this system is appropriate. In MVC architecture, the application is segregated into three groups, namely, the "Model," the "View," and the "Controller." The "View" represents the User Interface (UI) and all elements that are viewed by the user. As much as possible, MVC architecture will keep the UI files separate from server-side language files, in this case, PHP source code files. The "Model" represents the core server-side files that model how the application runs. The "Controller" represents the interface between the View and the Model files. The Controller directs user requests to and from the appropriate Model files. This development environment assists developers in MVC development practices by separating the interface files from the library of server-side classes through the use of controller files. The details of how this architecture functions will be discussed in further detail later in this chapter.

The entire application is organized within a single folder titled "ContexTable." This folder represents the CVS repository that contains the application. The entire application (other than software such as the database and web server) is committed to a CVS repository to enable multiple developers to work on the project simultaneously without disrupting one another's source code and to assist on controlling application versioning. Within the root directory, there are two additional folders "conf" and "www." The "conf" directory contains configuration files for the web server as well as

for the PHP preprocessor. This assists the researchers in not having to worry about configuration alterations that are necessary in order to support the application. The details for installation and configuration of the application can be found in Appendix A. The "conf" directory also contains the schema for the MySQL database used by the application.

The "www" directory is the root directory for the web server and is where the actual application resides. Each directory that is part of the application infrastructure will be explained in greater detail through the remainder of this section.

The "etc" Directory

This directory contains any configuration files needed by the application. The "etc/conf.php" file is the only file required to be included by the base application. This file sets global configurations, handles session settings, and includes any other configuration or source files needed by the application. Although future developers may wish to add to this file, most of the configurations required in order to build a family-centric application have already been developed and are included in this file. The "etc/custom.php" file is where researchers may quickly edit configuration values for the application. For example, a researcher may choose to change whether or not the application ignores unknown operating system processes when learning behaviors. The researcher can make one quick edit to the "etc/custom.php" file and this will change the way the application functions without needing to alter any core source files, and without needing to restart any services. As researchers develop their applications, any settings or variables that are best suited to a global configuration should be set in the "etc/custom.php" file, and then defined system-wide in the "etc/conf.php" file. This

enables the application to stay organized while allowing for efficient system-wide changes to the application.

The "lib" Directory

This directory contains all of the core application files needed to run the system. Each file in this directory represents an Object Oriented class. Each class contains variables, methods, and functions that pertain to the object that the class represents. Any file that a researcher places in this directory that has a ".php" extension will automatically be included by the development environment and made available to the rest of the application. This allows future developers to create custom classes and scripts that simply need to be placed within this directory and the class will automatically be included in the application without requiring code modification or restarting of any services. All database interaction, learning mechanisms, decision processing, and other core functionality reside within classes in the "lib" directory. The development environment was architected so that a researcher may develop a new class, save it to a file (preferably with the same filename as the class name), and then automatically begin using that class within the application without needing further configuration or application adjustment. For example, if a future project makes use of a special database table for tracking Radio Frequency Identification (RFID) tags on a family member's clothing, the developer should create a PHP class that represents the database object, save the filename with a ".php" extension, and then move the file to the "lib" directory. This custom class will then be available for access throughout the entire application.

The "do" Directory

This is the directory where all "Controllers" reside. In a typical MVC architecture, the web browser directs actions to a controller, which then interacts with model files, which then communicate information back to a view where the user can interpret the response. This development environment is built in the same manner, where the user interfaces with html-based files. These files will make calls to certain actions that are contained within the "do" directory. These files then interact with the core application files located in the "lib" directory (see the previous section). By organizing the structure in this manner, the application becomes much more organized and can be easily scaled without cluttering the environment. The following is an example of how the "do" files interact with the system. This use case describes how the code modules interface with one another in the system. Although an actual demonstration is not provided here, the modular programming structure will become apparent through the other demonstrations performed in this research.

Sally (a user of the system) clicks on a link on an html page that is served from the "ui" directory (see the section titled "ui") or performs an action that is detected by the Contextual Environment in an attempt to launch ApplicationX. This link directs Sally (transparent to her) to the "do/launchApp.php" directory. This file then uses the core source files located in the "lib" directory to interact with the database, learn her behavior, compare it to previous behavior, and then return results based on that interaction. The "do/launchApp.php" then launches her requested application and redirects (again, transparent to Sally) to a typical view of the application that makes sense to her. In this

case, it might return her to the menu of applications after launching her requested program; it may also activate actuators in her environment (e.g. open a garage door).

As can be seen from the example, Sally only sees a typical html-based user interface or interacts with sensors and actuators, while the application performs actions and makes decisions based on the results of those actions. The "do" files act as the controllers for interaction between the user and the system.

By separating the interface from the action processes and logical processes, future developers may simultaneously work on different portions of the application without concern for interfering with another developer's work. For example, one researcher may be experimenting with two or three different interfaces, including, perhaps, a Macromedia Flash-based graphical interface. At the same time, another researcher may be enhancing the processing of sensors and actuators. As the first researcher is constantly making changes to the interface, the second researcher's progress is in no way hindered because the files that store the User Interface (UI) information and the files that store the sensor and actuator processing are completely separated. At no time should the UI researcher ever have a need to alter any core functionality of the application that may affect the researcher who is developing the sensor and actuator programming. This allows multiple researchers to work on a single project without damaging the effectiveness of the work.

The "phpmyadmin" Directory

This directory is not a core portion of the application, and is not necessary for the functioning of the development environment. This directory contains an interface to the database that allows one to view and manipulate the structure and content of the database

via a web browser. This has been included in the development environment primarily for demonstration and test purposes. Certain demonstrations in this research require viewing the data contained within the database, and PHPMyAdmin provides a browser-based method for accomplishing this; but this is not a necessary element in the development environment. Despite the fact that this is not critical to the development or execution of an application, it is being left as part of the development environment as a feature to simplify database development and testing for future researchers who may not be familiar with direct queries on a database.

The "ui" Directory

This directory is where the user interface for the application resides and represents the "View" portion of the MVC architecture. Future developers of the application should organize the project such that all user interface elements are stored within this directory, or more particularly, in one of its subdirectories. The architecture and detail for this portion of the development environment will be described in detail in the section of this document titled "Varying Levels of Computer Familiarity Among Users".

Dynamic Family Structure and Lifestyles

Although a family unit does not change on a frequent basis, the system needs to adapt to a changing family structure. For example, a new child may be adopted or born into the family; or an older child may leave to go off for college. In order to account for this, the database was structured such that a table in the database contains an entry for all family members, or other users, that are part of the system. The structure for the learning tables is such that it is not dependant on the presence of any single family member, or

even of any family members for that matter (to allow for future expansion of project purpose). Any family members that are present become "Actors" in the system. "Actors" here is used in the UML sense. Future developers may then create virtually any interface for editing family structure that simply needs to modify the database. This might be an automated detection system, or something simpler such as a family member registration form. In either case, the structure of the environment is such that adding a new family member into the database is all that is required for involving that family member in the environment. It should be noted that any external devices or recognition systems may still need to be configured or adjusted to allow for a changing family structure, but that effort is part of the external system.

It is important to note that this development environment was specifically designed to allow for the unique situations presented by a long-term family living experience. This includes the ability to modify the structure of the family, but more importantly, the ability to continually modify the way the end application reacts according to the family member's preferences or the context surrounding the family member's situation. All information relating to any given family member is stored within the relational database table that represents a user. This table can be easily modified to add or remove parameters that may be required by the system. For example, if the system requires that the weight of each family member be known, then a column that represents weight simply needs to be added into the database, and then subsequently added into the "lib/Actor.php" file, which contains the class that handles database interaction for the family member. This structure then allows the parameters of any family to be changed with minimal modification, and without disrupting other modules or

code classes within the application library. At any time, if that family member's weight changes, then the external system simply needs to update the weight value for that family member in the database. No source code changes are required in order to make these value adjustments. It is also extremely unobtrusive to edit the number of family members. If a child goes off to college, and no longer is a part of the system, the record that represents that family member simply needs to be deleted from the database. Future researchers may create a simple interface for making this database query. Once the record has been removed from the single database query, the system will automatically update and adjust to the fact that there is one less family member. The same is true for adding a new family member.

To demonstrate this process Figure 4.2 shows the structure of some of the database tables both before adding a family member and after adding a family member. First, the family member was manually added to the "All Family Members" list with an ID of "5". This ID is what the rest of the system uses to identify the family member. Any other attributes, such as name, weight, age or other information that an external system might use to identify the user can simply be added as a new field in the family table (known in the database as "Actor"). Some external device would then indicate in the "Current Family Members" table that the new family member has arrived, based on the unique database ID number that was assigned to that family member (5). This process was simulated by manually adding the new family member to the database table. Some action would then be performed, which in this case was launching a 3rd party application, and the learning algorithm automatically recognizes the presence of the new family member without requiring any modification of source code. The new family

member ID shows up in the "Learning ID" table. This can be seen by the presence of the ID (5) after the "A:" in the learning ID. The "A" in the learning ID record represents one or more actors in the system. In this case, the actors are the present family members, but that can be adapted to the needs of the application being developed. A more detailed description of the learning ID is provided in the Appendixes to this research. The learning ID is then used by the learning and recognition algorithms in order to determine the context of past and present situations. This demonstration shows the development environment's ability to modify the structure of the family without requiring changes to source code, and without requiring any services to be restarted or reconfigured. A complete description of the database, including a description of each table as well as the relationships between tables is provided in the Appendixes to this research.

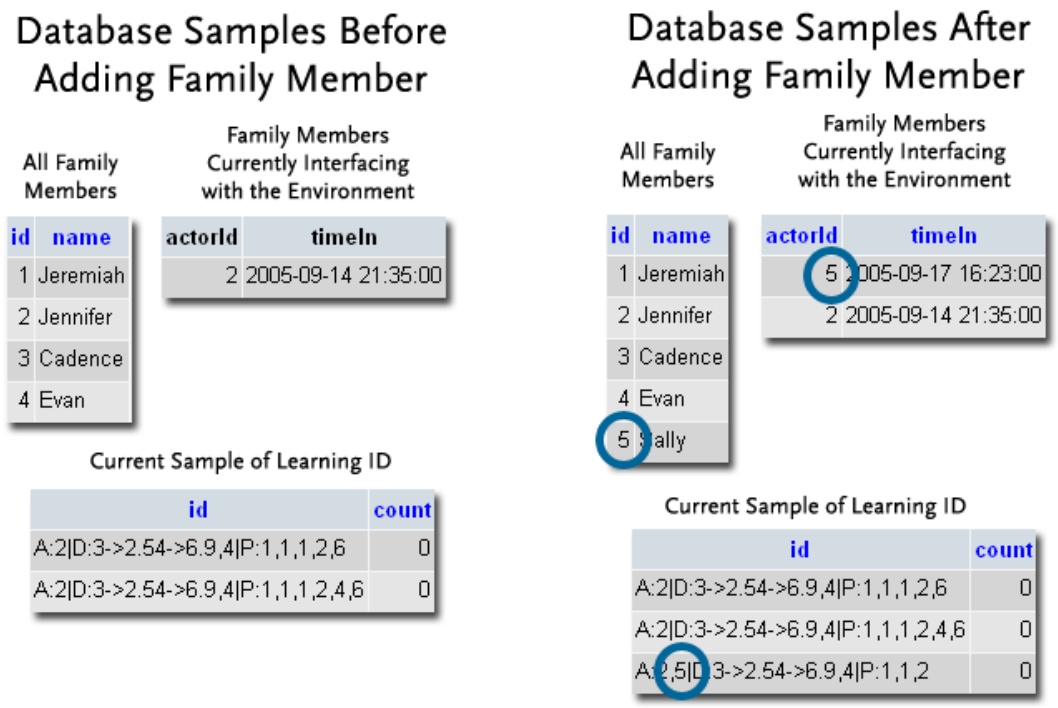


Figure 4.2 – Demonstration: Dynamic Family Structure

As another example of how the development environment is able to adapt to the long-term lifestyles of the family, let's examine how the development environment is used in the following use-case scenario. John, a 5 year old family member, wakes up each morning, and after getting dressed, approaches the computer to play his favorite educational game. After some time, the system learns this behavior, and the unique learning ID for this behavior is stored as a pattern in the environment's database. Each morning, as John approaches the computer, it uses the behavior recognition module and determines that John is coming to play his favorite educational game. The system then launches this game for John. Several months pass, and John continues playing his favorite game on a regular basis. However, one day John decides to try a new game. As he approaches the computer, the system launches the old game, which John promptly terminates. The development environment uses the limited two-way communication functionality to recognize that the game was terminated much earlier than expected, and the learning algorithm module updates the weight of the learning ID based on John's response. John then launches his new game, which triggers an event in the learning system and the context of the situation is recorded in the database. After one or two days of this occurring (depending on the specifics of the implemented learning algorithm), the system no longer launches the previously learned behavior, but instead recognizes the new pattern of behavior based on the new learning ID that has been recorded in the database. As John grows older, and becomes bored of the educational games, the system uses the same techniques and components described above to adjust to his constantly changing behavior. The created system utilized the structure of the development environment, the modularity of the learning algorithm and the recognition algorithm, as

well as the other components in the environment to keep a long-term perspective on the context of John's lifestyle and that of his family.

Browser Launching an External Application

One of the most unique challenges to developing family-centric applications is the challenge of building an application that can run via a web browser. As was discussed in previous chapters of this research, the web browser is the best option for developing an application that can be adapted to and used by the varying levels of computer comprehension that one might find in a typical family. It is critical that a family-centric application such as this be developed using a browser environment in order to provide familiarity to all levels of user comprehension. This poses a major limitation in that current web browsers limit the ability of the browser to interact with the host operating system. This makes communication between the browser and external processes or applications extremely difficult.

The development environment was designed to allow an application on the local machine to be launched by a web browser interface. This required the use of a 3rd-party plug-in called "LaunchinIE" (Mannaert, 2003). This plug-in is designed specifically for Internet Explorer on Windows, but other plug-ins are available for other browsers, or can be created by future researchers. The instructions for installing this plug-in can be found in the appendixes to this research.

In order to launch a specific application, it must first be manually entered into the database. Future development may involve the creation of a more automated method for adding applications to the system (such as a detection system that automatically adds repeatedly used applications to the database). Once the application has been entered into

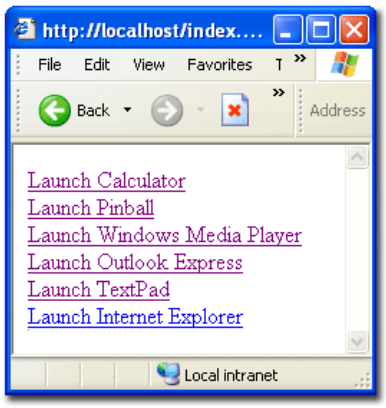
the database, it can instantly be launched by the project through the use of the development environment's built-in library of functions. The user interface can launch an application through a simple link, form, or other loading process that can make a call to an action URL that subsequently calls the "launchNow()" function that is included in the development environment. It should be noted that the interface for launching an application does not require modification to any server-side code, and can thus be completely redesigned based on future research, or the needs of the project being developed. For the most flexibility, it is suggested that future researchers utilize a redirect pattern, hidden frame, or other obfuscated method for launching the application. One such method is used in the following demonstration (Figure 4.3).

The demonstration in Figure 4.3 utilizes a hidden iFrame (a commonly used HTML element) to launch the external process without the user being aware of the hidden processes taking place. By simply clicking on a dynamically created URL (for which the interface is entirely customizable), a hidden request is made to the local server, which makes a call to the "launchNow()" function, specifying which application to launch. The application is then launched by the plug-in without needing to redirect the user to an alternate page.

Application Table in Database

id	name	desc	pName	runPath
1	Internet Explorer	Application used to browse the Internet	iexplore.exe	iexplore.exe
2	TextPad	Text Editor	textpad.exe	C:\\Program Files\\TextPad 4\\TextPad.exe
3	Outlook Express	Email Client	msimn.exe	C:\\Program Files\\Outlook Express\\msimn.exe
4	Windows Media Player	Digital Media Player	wmplayer.exe	C:\\Program Files\\Windows Media Player\\wmplayer....
5	Pinball	3-D Game of Pinball	pinball.exe	C:\\Program Files\\Windows NT\\Pinball\\PINBALL.EX...
6	Calculator	Calculator	calc.exe	calc.exe

Dynamic Application List in Browser



Function to Launch an Application

```
/**
* Launches the specified application
*
* This function is used to launch an application.
*
* @param    String path to application to launch
* @param    [String if specified, represents additional arguments
*/
function launchNow()
{
    /*  Determine whether or not there are arguments to pass    */
    if(func_num_args() > 1)
        $args = " ".func_get_arg(1);
    else $args = "";

    /*  Launch the application  */
    echo '
    <script language="JavaScript">
        obj = new ActiveXObject("LaunchinIE.Launch");
        obj.LaunchApplication(\''.$this->runPath.$args.'\');
    </script>';
    flush();
}//end function
```

Calculator Successfully Launches After Clicking on Link in Browser

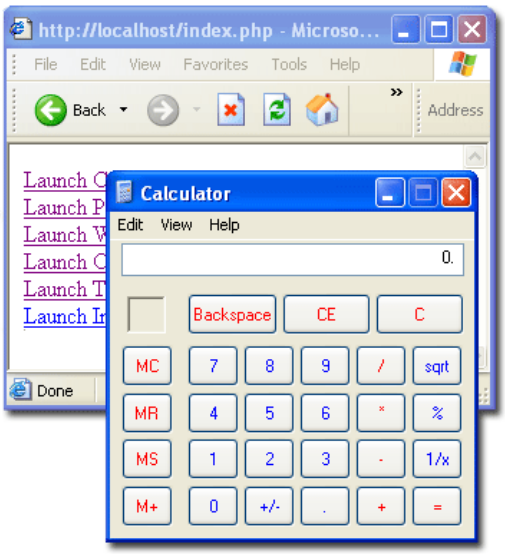


Figure 4.3 - Demonstration: Browser Launching External Application

Browser Terminating an External Application

In addition to the restrictions that browsers place on launching an external application, there are similar restrictions in terminating an external application. Rather than create a way for the browser to terminate an application, special functions were written utilizing server-side programming that can terminate a process on the local machine. Because the web server resides on the same machine as the developed application, server-side code may be utilized for terminating a process rather than dealing with the complexities involved in creating a browser plug-in to accomplish the same task. In order to terminate an application, the development environment must interface with the operating system using commands native to the operating system. In this implementation the commands were written for Windows XP Professional Edition. However, the "etc/Commands.php" file contains all of the native commands used by the application and future researchers can replace this file with the native commands used on another operating system without having to rewrite any of the core functionality involved in terminating a process. The following demonstration utilizes user interaction with the browser to terminate an application on the machine; however, it should be noted that the development environment can also integrate with an external process to accomplish the same task. If future researchers utilize an external device or system to trigger the termination of an application, the external system would simply need to make an HTTP GET request to the following URL: /do/endProcess.php?pid=[processed] (where [processed] is the process ID or name as listed in the task list. Figure 4.4 demonstrates the development environment's ability to terminate an application running on the local machine.

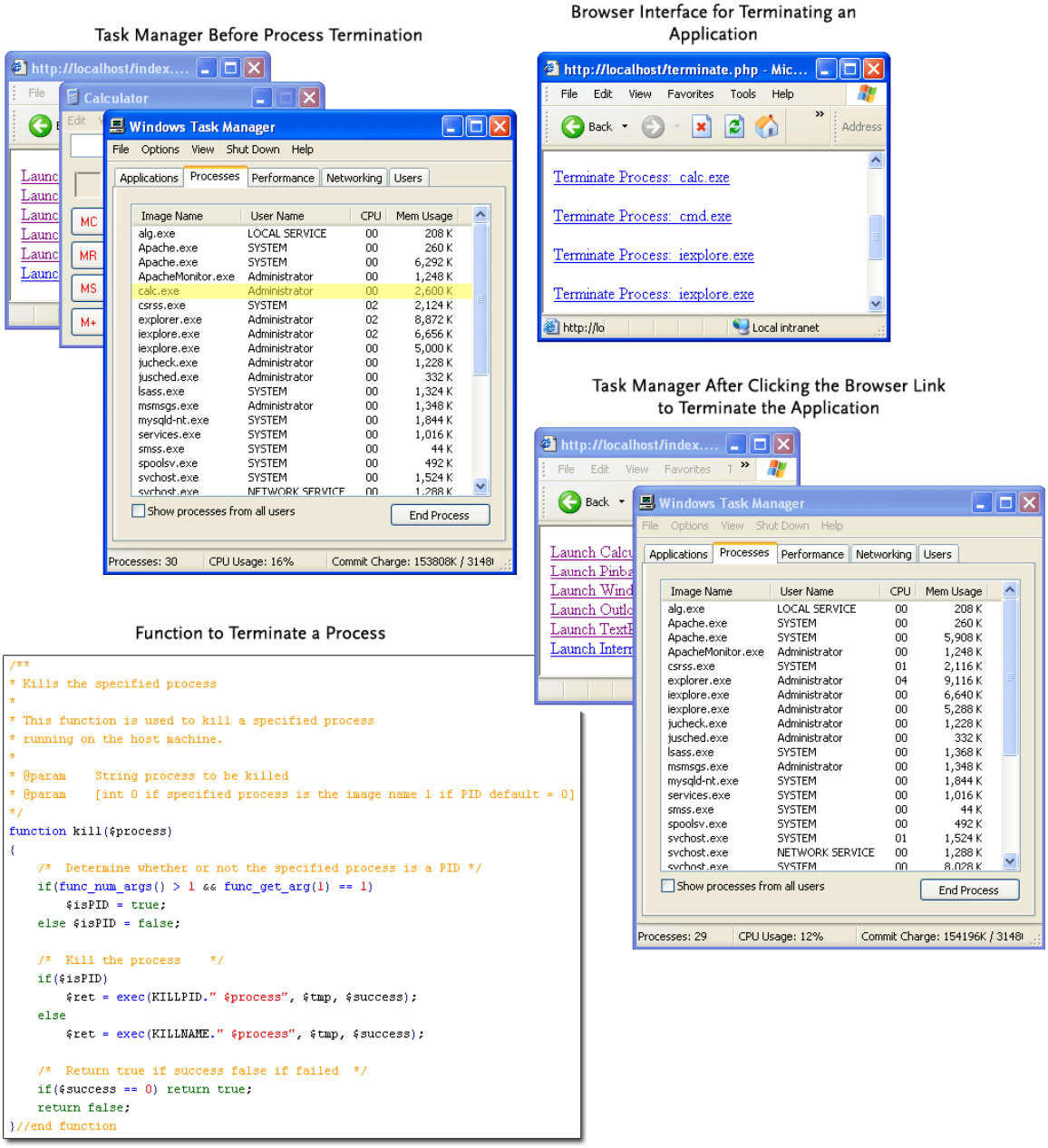


Figure 4.4 - Demonstration: Browser Terminating External Application

Modularity of Learning Algorithm

One of the most critical elements to implementing a context-aware system is the learning algorithm that is used. The scope of this research does not include creating or

refining a learning algorithm for use in the ContexTable project. However, the development environment must take into consideration the fact that the learning algorithm must be extremely modular. It is expected that future researchers will experiment with several different algorithms in an attempt to create the best algorithm for use in the ContexTable project. Furthermore, because this development environment is designed for multiple projects in a home-centric, context-aware environment, the learning algorithm may be extremely different from one project to another.

The development environment was architected so that the learning algorithm is implemented as an object within a class called "Brain." The "Brain" class can be instantiated and used to "learn" behaviors and output a result. The class uses other libraries to interact with the database. All processing for the learning and recognition (which will be discussed later) take place in the Brain class. The primary learning algorithm is contained within a function of the Brain class called "learn". This class is not called directly, but rather, a "stimulus" function indirectly calls this class after taking a snapshot of the current environment. By doing this, when an event occurs, or some other trigger occurs, the stimulus function is called, and the environment's current snapshot becomes the input(s) for the learning algorithm. It then modifies the database depending on the resulting output. To demonstrate the modularity of the learning algorithm, two different algorithms were written. Both are very simple, custom-made algorithms that exist only to serve the purpose of demonstrating the learning modularity. The first algorithm watches for a behavior to occur at least two times, after which point it considers it a pattern. It then gives the pattern a weight with both an upper and lower threshold. If the pattern occurs again, then depending on the weight, it will either execute

the pattern, or ask the user if it should be executed. Depending on the response, it will increment or decrement the weight. Once the weight reaches the upper threshold, the learning algorithm will automatically execute the desired output, without prompting the user. The second algorithm watches for a behavior to occur at least three times. After it seems the behavior occur three times, it considers it a pattern. Once it becomes a pattern, whenever the behavior is recognized again, the output will automatically execute without prompting the user.

The current development environment assumes that the learning algorithm is written in PHP and contained within the Brain class. If future researchers decide to utilize an external learning system, then the development environment may be modified to interact with an external system by simply replacing the code within the Brain class with the code needed to interface with the external process. This then keeps future developers from having to modify any portion of the development environment outside of the Brain class, thus eliminating the possibility of adversely affecting other portions of the application. This external process can then interact directly with the database in order to accomplish the same tasks as outlined in the following demonstration.

Figure 4.5, and Figure 4.6 show how the database is altered based on the two different algorithms. The important part of this demonstration is not the actual data in the database, but rather, it is the fact that the learning algorithm can be completely replaced without needing to alter the infrastructure of the application, or any source code outside of the Brain class. The source code for the Brain class, including the two learning algorithms can be found in the appendixes of this research.

Learning Algorithm 1

Both the behavior and pattern tables begin empty

After the first behavior occurs, the pattern table remains empty, but the behavior table has the following entry

id	count
A:2,5|D:3->2.54->6.9,4|P:1,1	0

After the behavior is recognized a second time, it is removed from the behavior table and is inserted into the pattern table starting at the lower threshold for weight.

id	weight	outputId
A:2,5|D:3->2.54->6.9,4|P:1,1	3	6

After the pattern is recognized again, the weight, and the user approves (simulated), the weight is incremented and the

id	weight	outputId
A:2,5|D:3->2.54->6.9,4|P:1,1	4	6

The same process is repeated again, and the weight is again incremented

id	weight	outputId
A:2,5|D:3->2.54->6.9,4|P:1,1	5	6

Figure 4.5 - Demonstration: Modularity of Learning Algorithm (Algorithm 1)

Learning Algorithm 2

Both the behavior table and the pattern table begin empty

After the first behavior occurs, it is inserted into the behavior table with a count of 1

id	count
A:2,5|D:3->2.54->6.9,4|P:1,1	1

After the behavior occurs again, the count is incremented

id	count
A:2,5|D:3->2.54->6.9,4|P:1,1	2

The count is again incremented when the behavior occurs a third time

id	count
A:2,5|D:3->2.54->6.9,4|P:1,1	3

When the behavior occurs a fourth time, it is removed from the behavior table and is inserted into the pattern table and will exectue every time it occurs after this point

id	weight	outputId
A:2,5|D:3->2.54->6.9,4|P:1,1	0	6

Figure 4.6 - Demonstration: Modularity of Learning Algorithm (Algorithm 2)

Behavior Recognition

A major part of the learning algorithm is the ability to recognize a behavior or pattern. For the examples previously shown, the pattern recognition was based simply on an exact duplicate scenario. Whenever the exact same scenario (as defined by the database learning ID) occurs, it is matched as a behavior or pattern. In a real-world situation, this type of behavior matching is most likely not sufficient. It is likely that a

behavior that should be learned could easily occur when circumstances are vastly different. For this reason, each portion of the learning ID (i.e. actors, external devices, time, and other variables) must be weighted differently, and some sort of recognition algorithm is necessary. This might be a statistical regression algorithm, a neural network, fuzzy logic, or some other form of recognition algorithm. It is not within the scope of this research to determine the best type of recognition algorithm to use. The development environment was architected to provide maximum flexibility in the recognition process. While the learning calculations and decision-making processes take place in the Brain class, two separate classes have been created that provide similar modularity for the recognition algorithms. The first class corresponds with the behavior table and is called "Path." The database table name is also called "Path." The second class is called "Pattern," as is the corresponding database table. The Path and Pattern classes are used to interact with those corresponding tables in the database, as well as perform functions relating to the respective objects. Each of these classes has a special function called "exists" which takes a single parameter which is a string representing the learning ID, and passes back a 'true' or 'false' based on whether or not the function thinks that behavior or pattern already exists. Currently, the function simply checks for an exact match in the database and returns true if one exists, and false if one does not. While this is adequate for the purposes of demonstrating the development environment's flexibility, it should be replaced by future researchers with a more complex algorithm that identifies whether or not a given behavior or pattern has been recognized previously.

Because the organization of the recognition algorithms are structured the same as the Brain class and its learning function, no demonstration of this modularity is necessary

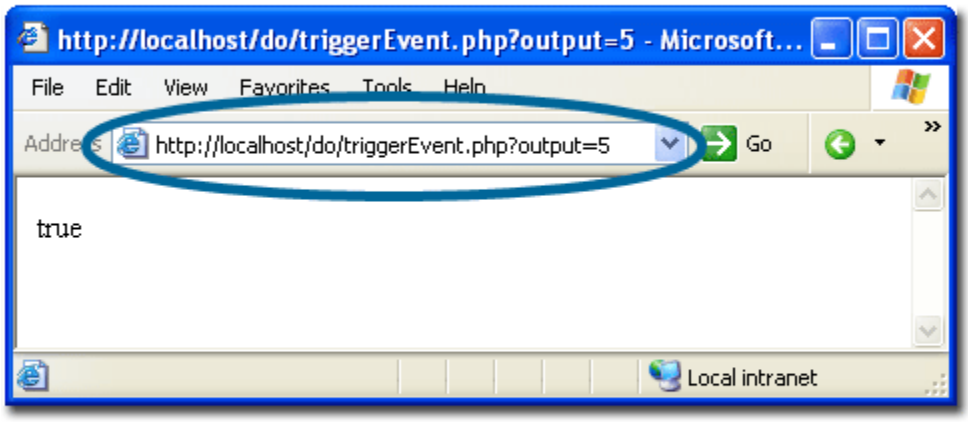
for the purposes of this research. It is sufficient to note that the recognition algorithms have been built in a modular fashion so that future researchers may further expound the capabilities of those functions.

Responding to Events and Sequences of Events

The ability to respond to events that are initiated through the browser has already been demonstrated through the results of the sections labeled "Browser Launching an External Application" and "Modularity of Learning Algorithm." The former demonstration shows how the user can initiate events in the browser, while the latter demonstrates the effect that this has on the database when controlled by the learning algorithm.

To demonstrate how third-party systems and applications outside of the browser can initiate events, a simple web service was created. This web service allows the development environment to be flexible enough to allow any third party system or application with HTTP GET capabilities to interact with the system. In order for a 3rd party system to trigger an event, it needs to be able to specify an output and then trigger the stimulus function that was described in a previous section. To accomplish this, the system simply needs to make an HTTP GET request to the "do/triggerEvent.php" file with a parameter "output" where the value of that parameter is the output ID as found in the database. For the purposes of this research, the various outputs must be manually entered into the database. Future researchers may wish to create a more automated interface for entering output options into the database. Figure 4.7 demonstrates how an event can be triggered via an HTTP GET request to the URL described previously, thus allowing external systems to interact with the environment via a standardized protocol.

Before triggering an event the behavior table is empty. The event is then triggered via a standard HTTP GET request (which can be made by any system with HTTP capability).



After triggering the event, the behavior table has been updated.

id	count
A:2,5|D:3->2.54->6.9,4|P:1,1	1

Figure 4.7 - Demonstration: Responding to Events

Having demonstrated this capability, it is important to note how the development environment is architected to handle event sequences. Often times, a single event may not indicate a pattern of behavior, but several events executed in a certain sequence may indicate a pattern. To handle this type of occurrence, the learning ID that is stored in the database is flexible to allow for multiple variables, of various types. One of these variables can be an event sequence. This can be added into the learning ID string as a comma-separated list of events that indicate a pattern. The learning algorithm, and in particular, the recognition algorithm would need to be refined to appropriately handle event sequencing, but the infrastructure for this is already provided by the development environment.

As an example of how sequences of events are handled in the system, the following hypothetical situation is provided. Each Monday evening, the family gathers together and spends the evening doing family activities. The family holds this as a regular event, and although the activities they perform may differ each time, there is a regular sequence of events that the family follows. The family gathers together and begins by pulling up an archive of favorite poems, short stories, or some other inspiring text. After reading one of the poems, the family then shares with one another the activities and events that occurred in that individual's life over the last week. Upon completing this conversation, the family pulls up an interactive family game from the computer and enjoys the activity together. Once the game has ended, the family then pulls up a scheduling program and proceeds to plan the events of the coming week, after which the family evening concludes and the family members return to their individual activities. Each week the family follows the same basic pattern, perhaps playing different games, or reading from different selections of poetry.

As the weeks pass, the Contextual Environment recognizes that the family is present, and the learning algorithm picks up on this regular, weekly behavior. As the evening progresses, the recognition algorithm is composed in such a way that it recognizes the regular pattern of events, as opposed to recognizing the singular events that take place. It recognizes that each Monday evening, once the family has gathered, they retrieve a specific document, followed by several minutes of conversation, after which the family entertainment system is activated, followed by the execution of a scheduling application. The learning algorithm records these sequences of events in the database as a separate learning ID. The learning ID uses a regular expression of "E:"

followed by a comma-separated list of the IDs for each of the other events (the actual expression is subjective, depending on how the learning algorithm is implemented). In the future, the recognition algorithm will recognize the execution of this sequence, and can retrieve the appropriate learning ID from the database, executing its predefined output. If the family's pattern is too irregular to recognize this as a sequence of events, certain sensors in the system may still be used alongside with the learning and recognition algorithms to trigger each event separately. The development environment provides the supporting infrastructure necessary to accomplish the recognition of events and sequences of events in the manner previously described.

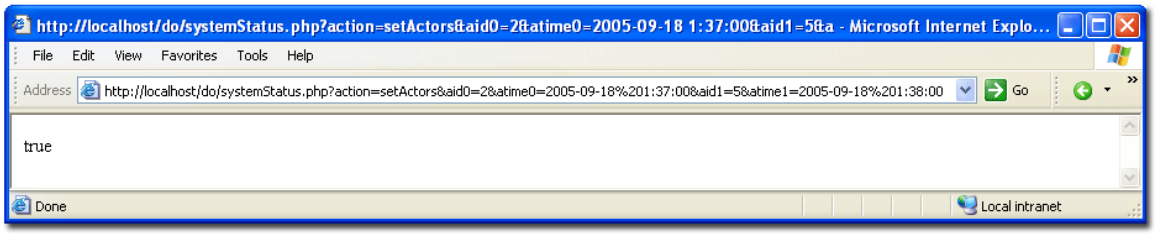
Ability to Receive Input from External System

Although this may seem similar to the previous section on "Responding to Events and Sequences of Events," it differs in one primary aspect—modifying the current status of the system. The external, context-aware system must be able to inform the application of who is interacting with the system as well as what devices, sensors, actuators, etc are available, and with what values. For this reason, a MySQL database was selected to hold information concerning the current status of the system. This would allow future researchers to create a middle-ware layer that can reside between the MySQL database and the external system in order to keep the application aware of what is happening in the "real world." A demonstration of how this direct manipulation of the database affects the system was previously provided in the section titled "Dynamic Family Structure and Lifestyles." When the new family member was added to the family table, a simulation was completed of an external system indicating that the new family member was interacting with the system. Subsequently, this family member's ID was included in the

resulting behavior database. An external system can directly modify the database tables that correspond with the current status of the family and any devices.

In addition to directly modifying the database, a web service was developed as part of the development environment to allow external systems to use a standard HTTP GET request to update the status of the actors and devices. Figure 4.8 demonstrates how the web service can be used to update the status of the devices and actors. Additional functions and actions can be performed through this web service, although they are not shown in this demonstration.

The URL shown below was requested in the browser to set the current actors



After requesting the previous URL, the following values were recorded in the current actor table

actorId	timeIn
2	2005-09-18 01:37:00
5	2005-09-18 01:38:00

The following URL was requested in the browser to set the current devices and their values

http://localhost/do/systemStatus.php?action=setDevices&did0=2&dtime0=2005-09-18%201:37:00&did1=5&dtime1=2005-09-18%201:38:00&dval00=1.1&dval01=1.2&dval02=1.3&dval10=2.1&dval11=2.2&dval12=2.3

After requesting the previous URL, the following values were recorded in the current device table

deviceId	timeIn	val1	val2	val3
2	2005-09-18 01:37:00	1.1	1.2	1.3
5	2005-09-18 01:38:00	2.1	2.2	2.3

Figure 4.8 - Demonstration: Ability to Receive Input From External System

Family-Friendly Interface

The development environment must take into account that fact that there will be family members of various competence levels and preferences interacting with the system. This is one of the most unique challenges that face the development of family-centric systems. As outlined in previous chapters, the web browser presents the most flexible and most familiar interface for developing applications suited for family use. Although it is not within the scope of this research to determine what interface designs are best, and how they should be implemented, the development environment must have the ability to utilize multiple interfaces. Projects developed through this environment will likely require various interfaces that can be presented based on the comprehension level of the user. Because it is necessary for the environment to support multiple interfaces, it is critical that the UI be clearly separated from the workings of the application. Otherwise the development of a family-centric application would be greatly hindered by the complexity involved in altering or swapping out interfaces. For this reason, the MVC method for architecting an application was designed into the architecture of the development environment. This keeps the interface separated from the functionality and programming that drives the application. In order to accomplish this, a "ui" directory was created within the root web directory. Within this directory resides an individual folder for each interface that is available to the application. Based on the URL requested, the interface can be dynamically selected. Future researchers may use this same methodology provided by the development environment to dynamically detect the interface based on the presence of specific users, and redirect the URL accordingly. This

allows future researchers to experiment with various interfaces as well as present different users with an interface dynamically suited to them.

As an example, a future project created in this development environment may implement a simple, Macromedia Flash-based, animated interface that is suited for young children in the family. It may simultaneously run a more complex, interactive, and more vibrant interface that is suited to teenagers within the family unit. It may also implement a more professional or more passive-looking interface that is suited to the parents of the family. Furthermore, it may have another interface that utilizes nothing but large text with limited icons for use by an elderly member of the family. As the system recognizes who is using the system, it simply redirects the URL to the appropriate interface, and the application then runs via the interface that is appropriate or preferred by that user. If a researcher decides to implement yet another interface geared towards young adults, the researcher simply needs to create the interface files and place them in a new template folder within the "ui" directory. Some application programming may be necessary in order to integrate the logic for when this interface is used as opposed to one of the others; however, the primary functionality of the application is in no way disturbed by the presence of the new interface. Additionally, the UI developer does not need to have an intricate understanding of how the application functions, but only needs to know the appropriate code elements required to interface with the server-side application.

To demonstrate how the development environment is able to have multiple interfaces that utilize the same back-end functionality, two distinct interfaces were created to accomplish the same task – launch an application via the browser. The first interface is a text-only interface, while the second is an image-only interface. Both

interfaces were able to connect to the same back-end without any modification and perform the task of launching an application from the browser. Figure 4.9 shows both of the interfaces that were used in the demonstration.

Summary

Each of the demonstrations described above show how the development environment was created in a way that allows future researchers to develop family-assistive technology that is context-aware. In particular, the development environment is capable of displaying multiple interfaces depending on the family member(s) present. Future researchers may investigate and experiment with the types of interfaces that should be integrated with family-assistive technology. The development environment accommodates the long-term needs of a family environment, including the ability to alter the structure of the family unit, as well as adapt to the changing lifestyles of each family member. The development environment separates each major component in future applications in a way that allows multiple researchers to work on different portions of the project simultaneously, with minimal communication and cooperation between researchers. The previous demonstrations prove the successful implementation of each component in the environment. These demonstrations also successfully prove the interfaces between these components. The following paragraphs summarize the interfaces between the environment's components and the demonstrations that prove the successful implementation of these interfaces.

Text-Based Interface

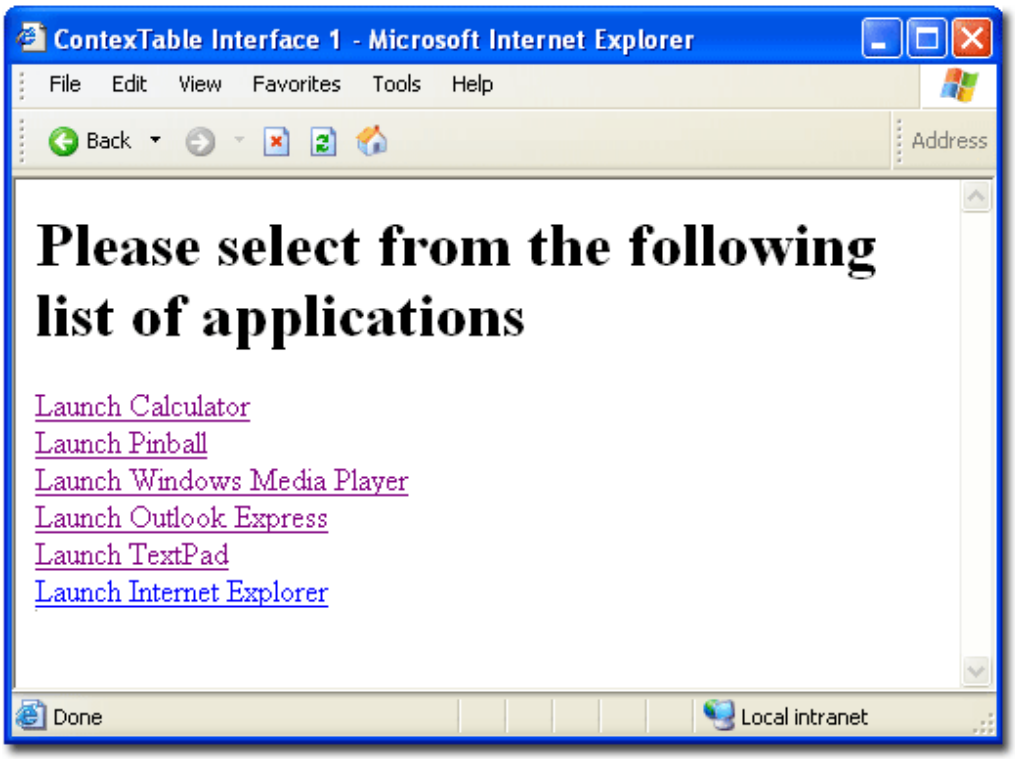


Image-Based Interface

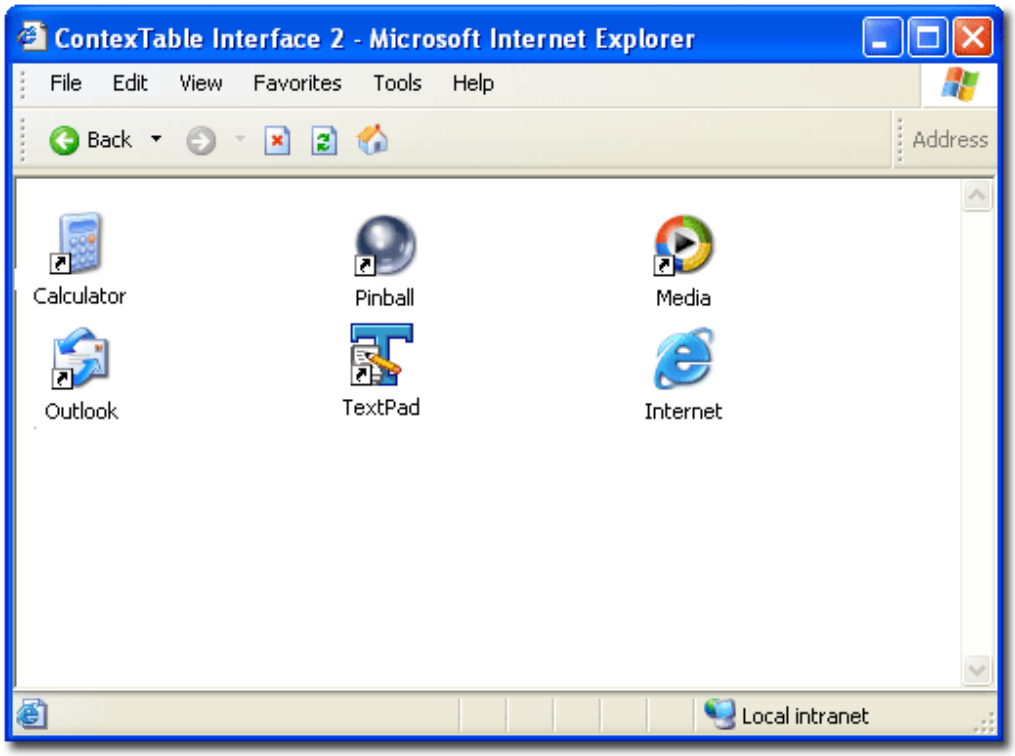


Figure 4.9 - Demonstration: Family-Friendly Interface

The interface between the family and the browser has been shown through each of the demonstrations. Each demonstration involved interaction between a family member and the browser interface.

The interface between the family and the Contextual Environment is not included as part of this development environment. Each application built from this development environment may require vast differences in this interface and thus it is not within the scope of this research to attempt an enumeration of all possible interfaces. However, several examples of this interface were provided through the use cases that accompanied the demonstrations.

The interface between the Contextual Environment and the browser was demonstrated in the "Ability to Receive Input from External System" demonstration. Although the Contextual Environment was simulated, the demonstration shows how the Contextual Environment is able to interface with the browser through a web service.

The interface between the Contextual Environment and the database was established in the "Ability to Receive Input from External System." Once again, the Contextual Environment was simulated, but in this and other demonstrations, it was verified that the database can be accessed directly by the Contextual Environment.

The browser and external applications interface was demonstrated through the "Browser Launching an External Application" and the "Browser Terminating an External Application" demonstrations. This interface also became part of several other demonstrations.

The browser and web server interface was used throughout most of the demonstrations. All of the demonstrations that utilized the browser showed successful communication between the browser and the web server.

The interface between the web server and the master control was successfully implemented in the following demonstrations: “Dynamic Family Structure and Lifestyles,” “Modularity of Learning Algorithm,” “Responding to Events and Sequences of Events,” “Ability to Receive Input from External System,” and “Family-Friendly Interface.”

Lastly, the interface between the master control and the database was proven through each of the following demonstrations: “Dynamic Family Structure and Lifestyles,” “Modularity of Learning Algorithm,” “Responding to Events and Sequences of Events,” and “Ability to Receive Input from External System.”

The completion of each component and its interfaces yields the successful creation of a development environment suited to creating family-assistive technology that is context-aware.

CHAPTER 5: SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

Summary

Context-aware systems seek to lessen the burden that is placed upon users interacting with the system. A context-aware system attempts to model the system's behavior around that of the user, so that the user is not required to interface directly with the technology as much as with a traditional system. Despite the advantages of context-aware systems, they have not yet become prevalent in family living environments. An important reason for this is the difficulties and challenges presented by creating a family-centric system that is context-aware. The difficulties that confront this area can be divided into three categories: *Issues of Context, Issues of Computation and Learning*, and *Technology* Limitations. Within these three categories, there are many problems that face development of a context-aware, family-assistive system. While some of these challenges also hinder the development of context-aware systems in general, there are three specific challenges that arise when designing such a system that is centered around the family. These include: the wide range of family member comprehension levels, the modification of the family structure, and the changing lifestyles of family members.

Research in this area has been seriously hindered by the lack of a stable, yet flexible development environment that confronts each of these challenges in a way that overcomes the challenge. Through the creation of a development environment that

overcomes these challenges, particularly those suited specifically to the creation of family-assistive, context-aware technology, future research and development may increase and progress more quickly. By providing developers and researchers with an environment that solves these specific challenges, it frees the researchers such that energy and efforts can be focused on creating and implementing systems as opposed to spending time confronting these basic challenges.

This research involved the architecture and deployment of a development environment suited specifically to family-centric, context aware systems. Each necessary component in the development environment was outlined and simple demonstrations were created to illustrate the successful completion of each component. The interfaces between these components were also enumerated. Communication across these interfaces was proven through the same demonstrations. By proving the functionality of the components as well as how they interact and interface with the rest of the system, a complete, custom-built development system is presented. This development environment can now be used to create future context-aware systems for family-assisted living. Having overcome the difficult challenges that are specific to research in this area, and having provided an environment and method for future research, it is expected that developers and researchers will utilize this development environment to build multiple systems in the future.

Conclusions

Through a variety of demonstrations, the development environment was shown to potentially meet the requirements for future development of a family-centric, context aware system. The major challenges in developing such systems have been accounted for

through this environment, thus allowing researchers to focus on the development of an actual system without being overwhelmed by some of the major problems that such a system presents.

Of these problems, one of the most difficult requirements to meet was the ability to support a web browser based user interface. Due to security restrictions imposed upon web browsers, there were many challenges presented. However, each challenge was overcome through various methods that can be quickly and easily deployed in a family environment. Demonstrations for how to accomplish this were provided. The framework for developing and programming in the environment has been created and documented. Sample integration scripts have also been created.

The development environment fully supports multiple, dynamic interfaces that can be customized to the needs of individual family members, or types of users. This will allow future developers to experiment with multiple types of interfaces in an effort to find those that are best suited to a variety of family members.

The development environment provides the infrastructure necessary to support a changing family structure as well as the need to adapt to the constantly changing lifestyle of the family and its individual members. This is critical in implementing long-term solutions for assisting families. Not only will the structure of a family change, but the habits, preferences, and lifestyles of each family member, as well the family as a whole will change over time. The development environment has provided a methodology that will allow future developers to build upon this foundation to supply a versatile application that meets these long-term, dynamic needs.

In conclusion, based on the criteria for successful completion of this research, the development environment has met each of the requirements outlined in this document and can now be utilized by future researchers for development in the area of family-centric, context-aware systems such as the ContexTable project (Hoopes, 2004) and other future projects. This includes having the flexibility for different research approaches, the capacity to segregate projects into distinct development groups (such as interface, learning, sensors, etc.), the future portability to various platforms, and the coherent structure that specifically supports the needs of family-centric technology.

Although this research has not directly interacted with or affected the lifestyles of families, this research facilitates development in this area that will benefit family living. Future research in this area will be streamlined through the use of this environment. The long-term outcome of this research is to assist the development of systems designed to strengthen family relationships, facilitate better family communication, and assist families with daily living.

Recommendations

With the creation of the development environment, there is broad room for future research and development in the area of family-centric, context-aware computing. As this research was completed, several suggestions for future research have arisen.

Although documentation for installing and configuring the development environment is included in the appendixes of this document, it is suggested that future developers create an install script or install wizard that speeds the time required for installing and configuring not only the development environment, but the end-user application.

It is suggested that future researchers build an interface or system that allows for the addition/removal of family members. The development environment includes an API for accomplishing this interaction, but an interface other than the PHPMyAdmin system was not implemented. A custom interface for this task would ease the configuration required to adjust the family structure. This may include an automated system that detects the new presence or prolonged absence of family members, and adjusts the database accordingly.

It was not within the scope of this research to create a cross-browser or cross-platform compatible system; however, as the development environment was being created, cross-platform compatibility was always considered. Any platform specific commands or programming was separated into configuration files that can be easily replaced with those required by another system. However, this was not tested extensively and thus future researchers should expand on the ability to port the development environment and resulting applications to systems other than Internet Explorer on a Windows PC machine. Specifically, the native commands file which was created to interface with the operating system and the plug-in used to launch applications in Internet Explorer should be developed and ported to alternate platforms.

Although a simple web service was created to allow external devices to interface with the system, it is recommended that future researchers expand the capabilities of this web service to provide further flexibility. It is also recommended that the web service be converted to the SOAP protocol (W3C, 2003), which is currently an accepted standard for web service communication. The current web service uses HTTP GET to initiate actions or interface with the system, but receives little response other than a “true” or

"false" indication of whether or not the command was successful. An expanded web service may also serve to port an interface to alternate devices such as a handheld device, cell phone, or remote system.

The development environment was specifically designed to allow extreme flexibility and modularity concerning the learning algorithm and behavior recognition algorithm. Although simple algorithms were created to accompany the development environment, it is highly recommended, if not necessary, that future researches experiment with a variety of algorithms to determine the most appropriate type of learning and recognition system to use. The algorithms selected for the application should most likely differ from one type of application to another, thus a single solution for the development environment may not be plausible.

This research did not extend into the realm of interfacing with external devices except to show that the web service and database architecture make this possible. Although other researchers are currently developing a system to interface specifically with the ContexTable project, future research in other applications besides the ContexTable project are also highly recommended. External devices that may not be used in the ContexTable project can also interface with the development environment, and thus further research into this area is strongly encouraged.

Lastly, the development environment was architected to be extremely flexible in terms of the user interface that is attached to the system. Moreover, it was designed to handle multiple interfaces. It is also possible to create a dynamic interface that changes from one user to another, or even evolves as the user interacts with the system. The scope of this research did not include experimentation with user interfaces with the

purpose of determining what type of interface(s) should be used with the system. Future researchers should determine types of appropriate interfaces and perform usability testing while creating one or more interfaces that may be implemented with the system. The interface will also largely depend on the application. Therefore, the ContexTable project may implement an entirely different interface than other family-centric, context-aware systems that may be developed in the future.

REFERENCES

Anhalt, J., Smailagic, A., Siewiorek, D.P., Gemperle, F., Salber, D., Weber, S., Beck, J., Jennings, J. (2001) Toward context-aware computing: Experiences and lessons IEEE Intelligent Systems and Their Applications, v 16, n 3, p 38-46

Binemann-Zdanowicz A., Kaschek, R., Schewe, K., Thalheim, B., (2004) Context-Aware Web Information Systems. Proceedings of the first Asian-Pacific conference on Conceptual modeling – Volume 31.

Capra, L., Emmerich, W, Mascolo, C. (2003) CARISMA: Context-Aware Reflective middleware System for Mobile Applications, IEEE Transactions on Software Engineering, v 29, n 10, pp 929-945

Capra, R., Perez-Quinones, M., Ramakrishnan, N., (2001) Webcontext: Remote Access to Shared Context. Proceedings of the 2001 workshop on Perceptive user interfaces.

Chan, A. T.S., & Chuang, S., (2003) MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing, Source: IEEE Transactions on Software Engineering, v 29, n 12, pp 1072-1085

Chesley, N. (2005) Using Information Technology to Manage Work and Family Life: Implications for Life Quality. Dissertation Abstracts International, A: The Humanities and Social Sciences, 2005, 65, 8, Feb, 3172-A

Dey, A.K. (2003) Understanding and Using Context, Future Computing Environments Group (submission).

Dey, A. K., Salber, D., Abowd, G. D., Futakawa, M. (1999) Conference assistant: combining context-awareness with wearable computing. International Symposium on Wearable Computers, Digest of Papers, p 21-28

Dunst, C. J., Hamby, D., Trivette, C. M., Raab, M., Bruder, M. B. (2002) Young Children's Participation in Everyday Family and Community Activity. Psychological Reports, Volume 91, pp 875-897

Farringdon, J., Moore, A. J., Tilbury, N., Church, J., Biemond, P. D. (1999) Wearable sensor badge & sensor jacket for context awareness, International Symposium on Wearable Computers, Digest of Papers, pp 107-113

Gieselmann, P., Denecke, M. (2003) Towards Multimodal Interaction with an Intelligent Room. Proceedings of the Eurospeech, Geneva.

Hansen, S., Fossum, T. (2005) Refactoring model-view-controller. Journal of Computing Sciences in Colleges. Consortium for Computing Sciences in Colleges, volume 21, issue 1, pp 120-129.

Hess, C. K., & Campbell, R.H. (2003) An application of a context-aware file system. Personal and Ubiquitous Computing, Volume 7 Issue 6

Hoopes, D. M. (2004) The ContexTable: Building and Testing an Intelligent, Context-Aware Kitchen Table

Hughes, R., Ebata, A. T., Dollahite, D. C. (1999) Family Life in the Information Age. Family Relations, Volume 48, No. 1

Jackson, M. (2005) The Limits of Connectivity: Technology and 21st-Century Life. From work-family balance to work-family interaction: Changing the metaphor. Lawrence Erlbaum Associates, pp 135-150.

Kraut, R., Patterson, M., Lundmark, V., Kiesler, S., Mukophadhyay, T., Sherlis, W. (1998). Internet Paradox: A Social Technology that Reduces Social Involvement and Psychological Well-Being. American Psychologist, 53, 1017-1031.

Lieberman, H., & Selker, T. (2000) Out of Context: Computer Systems that adapt to and learn from, context. IBM Systems Journal, 39, 617-632.

Maamar, Z., Kouadri, S., Yahyaoui, H., (2004) A Web Services Composition Approach Based on Software Agents and Context. Proceedings of the 2004 ACM symposium on Applied Computing.

Mandato, D., Kovacs, E, Hohl, F., Amir-Alikhani, H., (2002) CAMP: A context-aware mobile portal, IEEE Communications Magazine, v 40, n 1, p 90-97

Mannaerts, K. (2003). LaunchInIE. (Online), retrieved October 08, 2005. WhirlyWiryWeb.com. http://www.whirlywiryweb.com/q/%2Flaunchinie.asp

Moran, T.P. & Dourish, P. (2001) Introduction to This Special Issue on Context-Aware Computing, Human-Computer Interaction, Lawrence Erlbaum Associates, Inc., 16, 87-95.

Myer, S., Rakotonirainy, A. (2003). A survey of research on context-aware homes. Conferences in Research and Practice in Informaiton Technology Series; Vol. 34. Australian Computer Society, Inc, pp 159-168.

Perry, D. E., Kaiser, G. E. (1988) Models of Software Development Environments. Software Engineering – Proceedings of the 10th International Conference 11-15 April 1988, pp 60-68

Petrelli, D., Not. E., Strapparava, C., Stock, O., & Zancanaro, M. (2000) Modeling Context is Like Taking Pictures. Proceedings of the 2000 Conference on Human Factors in Computing Systems (ACM SIGCHI 2000).

PHP. (2005). Usage Stats for September 2005. (Online), October 25, 2005. http://www.php.net/usage.php

Pyne, R. A., Mugisa, E. K. (2004) Essential Elements of a Component-Based Development Environment for the Software Supermarket. Proceedings of the Mar 26-29 2004 Southeast Con (IEEE), pp 173-180

Rees, M. J. (2002) Evolving the Browser Towards a Standard User Interface Architecture. Australian Computer Science Communications, Volume 24, issue 4

Rogina, I., Schaaf, I. "Lecture and Presentation Tracking in an Intelligent Meeting Room," icmi, vol. 00, no. , p. 47, Fourth 2002.

Rosin, N. et al (2004) HomeOS: Context-Aware Home Connectivity. Proceedings of the International Conference on Wireless Networks, ICWN'04 - Proceedings of the International Conference on Pervasive Computing and Communications, PCC'04. CSREA Press, pp739-744.

Selker, T., Burleson, W., & Arroyo, E. (2002) E-Windshield: a study of using. ACM Conference on Human Factors and Computing Systems: Demonstrations (ACM SIGCHI 2002), 508-509.

Statistical Research, Inc. (1998) Technology in America: Tune in, log on, dial out. [Online] Available: http://www.sriresearch.com/pr062298.html [1998, November 24]

Sutschet, G. (2005) The CHIL Software Architecture. Fraunhofer IITM VACE Conference

Svanaes, D. (2001) Context-Aware Technology: A Phenomenological Perspective. Human Computer Interaction, Lawrence Erlbaum Associates, Inc. 16, 379-400.

Taylor, J. K. (2004) North American Markets for Industrial Electronic Monitors, Operator Interface Terminals, and Associated Software. Venture Development Corporation

W3C (2003). SOAP Specifications. (Online), retrieved October 08, 2005. World Wide Web Consortium. http://www.w3.org/TR/soap/

APPENDIXES

Appendix A: Development Environment Setup Steps

Start with fresh install of WinXP Pro on Virtual PC

Download TextPad for editing purposes from http://www.textpad.com

Download latest version of Apache for Windows from http://httpd.apache.org

Download latest version of MySQL for Windows from http://www.mysql.com

Download latest version of PHP for Windows from http://www.php.net

Download latest version of Tortoise CVS from http://www.tortoisecvs.org

Download latest version of LaunchinIE from http://www.whirlywiryweb.com/q/%2Flaunchinie.asp and follow the instructions on the site for installation. Set only one acceptable URL and the value should be http://localhost/. Be sure to include the trailing slash. The trailing slash will prevent browser hijacks from URLs such as localhost.bad-domain.com

Run the Apache installer and select default options, with these exceptions:

Network Domain = localdomain

Server Name = localhost

Any firewall warnings should be set to "unblock" for this and the remaining installers.

Run the MySQL installer with default options. Select to configure the server after install. Configure it with defaults except here:

Select “Server Machine”

Select “Non-transactional database only”

Select “Include Bin directory in windows path”

Set the root password to “contextable” (or another password of your choosing)

**Note, you may get a windows warning about the firewall. If so, add the “MySQL” service on port 3306 to the firewall exceptions.

Extract the downloaded php file into c:/php

Run the Tortoise CVS installer with default options.

Create the following directory “C:\cvsroot” and inside of it checkout the “ContexTable” module from the CVS server. At the time of this document, the CVSROOT is:

:pserver:jeremiah@10.10.10.30:/home/cvs

Edit the httpd.conf file and replace all text with the following line:

Include "C:\cvsroot\ContexTable\conf\httpd.conf"

Appendix B: Summary of Scope

This Appendix represents a comprehensive list of concepts, components, or items that may relate to this research, but were mentioned in this document as being outside of the scope of this research. This list should serve as a more comprehensive list of delimitations for this research, and may be utilized by future researchers desiring to expand upon this research.

This research does not include:

- Implementation of a stable or robust learning algorithm
- General two-way communication between a browser and external applications
- Logic for database cleanup methodology and scheduling
- Tests for cross-platform compatibility
- Demonstration of event sequencing
- Creation of scheduled tasks
- Demonstration of dynamic interfaces that change based on user preferences
- Creation of a completed family-centric, context-aware system
- Determining the most appropriate recognition algorithm
- Determining the most appropriate interface designs
- Exploration of security enforcement within the environment
- Enumeration of possible interfaces between the Contextual Environment and the family

Appendix C: Glossary of Terms

This appendix contains a glossary of the more technical terms used throughout this document.

ActiveX – Microsoft software components.

API (Application Program Interface) – A set of functions, methods, or calling conventions that define how an application can be interfaced with.

Browser – A program capable of reading and displaying Hypertext and interacting with the World Wide Web.

Context-Aware Computing – A form of computing where a system is aware of, or reacts to the context of the situation in which the system is run.

CVS (Concurrent Versioning System) – An open source solution for versioning control of source code and applications.

Development Environment – Software or system designed to assist in developing software for a particular language, system, or situation.

Family-Assistive Technology – Technology designed to enhance family living and assist families perform tasks more efficiently.

Family-Friendly – Used to describe a system or interface that is easy for all of the various members of a family to use.

HTTP (HyperText Transfer Protocol) – The communication protocol used for web browsing and many Internet applications.

MVC (Model-View-Control) – A specific software architecture that separates software into subsystems that represent the user view, the modeling logic, and the application control logic.

Object Oriented – A module of data that is self-contained within its associated code.

Open Source – Software that publicizes or freely distributes source code for the use and benefit of other developers.

Real-time – Refers to an application that performs specific tasks or conditions within a pre-defined time constraint.

Source Code – The textual form and original syntax of a program.

Ubiquitous Computing – Integrating computing into the environment.

UI (User Interface) – The portion of a system that a user interacts directly with.

Appendix D: Database Schema and Description

The database used in this application was a MySQL Relational database. The database consists of a minimum of 8 tables: *Pattern, Path, Process, Actor, Device, currActors, currDevices,* and *currProcesses.* The *Pattern* table represents contextual imprints that represent a known pattern. This utilizes the learningId as described in this research. The learningId provided as an example in the demonstrations utilized a syntax where parts of the context were delimited by a letter-code followed by a colon, followed by the information for that contextual component. Each contextual component was then delimited by a pipe, or "|" symbol. The example components used in this research were "A" representing family members, "D" representing devices, "P" representing 3rd party applications, and "E" representing events. Following this syntax, an example learningId, as implemented in this research, might be as follows:

A:2,5|D:1,2->3.5->4,43|P:23,34

The above learningId indicates that two family members were involved in the context (represented by ID #s 2 & 5); there were three devices involved in the context (1,2,43), with one of those devices registering two different values; there were two 3rd party applications running under this context (23,34).

The *Path* table also utilizes the learningId pattern. This table is used to store contextual situations that are not known to be patterns. It is a record, or history of past contexts used to learn behaviors in the future.

The *Actor* table represents the family members. The columns in this table should be modified to reflect the information that the system must know about each family member.

The *Device* table represents external sensors and actuators that the system must interact with. The columns in this table should be modified to store information about the devices that the system may need to know.

The *Process* table represents 3rd party applications that interact with the system. These columns include information about the task name (used for terminating processes) and the run path (used for executing processes).

The *currActors* table represents the family members who are currently interacting with the system. This table may also be modified to include the devices with which the family member is interacting. That information may also be stored in the *currDevices* table.

The *currDevices* table represents the external sensors and actuators that are currently active in the system. This table has several columns that may be used to store custom values that the sensors may register for use in understanding the context. Additional columns may be added as necessary to understand the context of a device.

The *currProcesses* table represents the processes or 3rd party applications that are currently running. Although the API for the source code also provides a method for retrieving this directly from the Operating System, this information is recommended to be stored in the database as well to assist external devices or processes in knowing what processes are currently running. Custom functions and methods are included in this API

for the efficient storage and synchronization of this table with the Operating System's running processes.

The following diagram provides the relational schema used by the databases provided with this development environment.

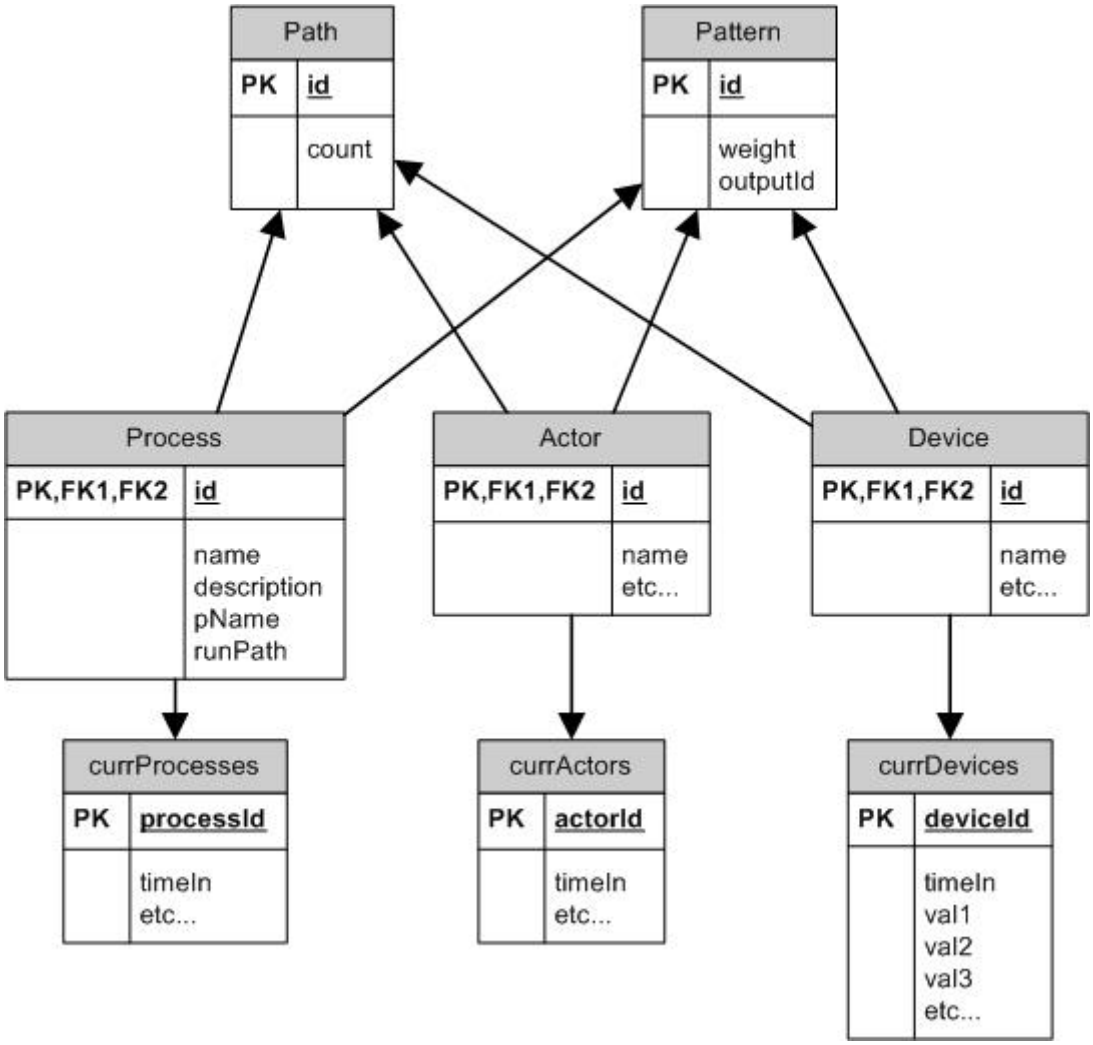


Figure 5.1 - Database Schema

Appendix E: Source Code Documentation

/etc/custom.php

Description

This file contains all of the configurations for the environment.

This file contains all of the configurations for the environment. These values should be set to suit the needs of the given application.

Documentation generated on Thu, 3 Nov 2005 21:03:35 -0700 by phpDocumentor 1.3.0RC3

/etc/conf.php

Description

Description | Includes | Constants

Includes

Description | *Includes* | Constants

include_once (**'custom.php'**) (line 14)

This file contains all of the global configurations for the environment.

This file contains all of the global configurations for the environment. If a certain value is likely to change based on the configuration, then the value should first be represented in the /etc/custom.php file so that quick and simple modifications can be made to the commonly changed environment settings.

include_once (**LIB.$file**) (line 70)

include_once (**'Commands.php'**) (line 73)

Constants

Description | Includes | *Constants*

DB = $database (line 36)

Database name

Database name

DB_HOST = $host (line 24)

Database Host

Database Host

DB_PASS = $passwd (line 32)

Database Password

Database Password

DB_TYPE = $db_type (line 40)

Database Type

Database Type
DB_USER = $user (line 28)

Database User

Database User
DEFAULT_TEMPLATE = $default_template (line 64)

Default UI to use

Default UI to use
LIB = ROOT_PATH."lib/" (line 52)

Path to the code library

Path to the code library
NUM_DEV_VALS = $num_device_values (line 56)

Number of device values in the database table

Number of device values in the database table
ROOT_PATH = $root (line 48)

Root Application Path

Root Application Path
ROOT_URL = $url (line 44)

Root URL

Root URL
UKP = $use_known_processes (line 60)

Whether or not to ignore unkown processes

Whether or not to ignore unkown processes
USER = $localUser (line 20)

Local System User

Local System User

Class Actor

Description

Description | Vars (details) | Methods (details)

AUTHOR/CONTENT-OWNER: Jeremiah K. COPYRIGHT: Jeremiah K. DISCLAIMER: This code may only be used with permission from the author DESCRIPTION: This class represents a "Actor" object. to be used with a "Actor" table in a SQL Database. contained within this class are for use with manipulating and retrieving information from the "Actor" table.

AUTHOR/CONTENT-OWNER: Jeremiah K. Jones COPYRIGHT: Jeremiah K. Jones 2005 DISCLAIMER: This code may only be used with permission from the author DESCRIPTION: This class represents a "Actor" object. The variables and methods are to be used with a "Actor" table in a SQL Database. The variables and functions contained within this class are for use with manipulating and retrieving information from the "Actor" table.

*Located in **/lib/Actor.php** (line **14**)*

Variable Summary

Description | *Vars* (details) | Methods (details)

mixed **$cols**
mixed **$id**
mixed **$name**

Method Summary

Description | Vars (details) | *Methods* (details)

Actor **Actor** ()
void **addCurrent** (*mixed* **$actorId**, *mixed* **$timeIn**)
void **clear** ()
void **clearCurrent** ()
void **exists** (*mixed* **$id**)
void **getCount** ()
void **getCurrent** ()
void **getId** ()
void **getList** ()
void **getName** ()
void **morph** (*mixed* **$id**)
void **newId** ()
void **remove** (*mixed* **$temp**)
void **save** ()
void **setCurrent** ()
void **setId** (*mixed* **$id**)
void **setName** (*mixed* **$name**)

Variables

Description | Vars (*details*) | Methods (details)

mixed **$cols** = "id, name" (line 21)
mixed **$id** (line 19)

The variable names should mirror the columns in the database.

The variable names should mirror the columns in the database.
mixed **$name** (line 20)

Methods

Description | Vars (details) Methods (*details*)

Constructor Actor (line 26)

This is the constructor

This is the constructor
Actor **Actor** ()
addCurrent (line 170)

Adds a current actor to the table

Adds a current actor to the table
void **addCurrent** (*mixed* **$actorId**, *mixed* **$timeIn**)
clear (line 58)

This function clears out the information for the current object, and deletes it from the database.

This function clears out the information for the current object, and deletes it from the database.
void **clear** ()
clearCurrent (line 131)

Clears out current Actors.

Clears out current Actors. If an id is specified, then it only clears that id
void **clearCurrent** ()
exists (line 97)

This function checks to see whether or not a specified object exists.

This function checks to see whether or not a specified object exists.
void **exists** (*mixed* **$id**)
getCount (line 47)

This class will return the total number of objects in the database

This class will return the total number of objects in the database
void **getCount** ()
getCurrent (line 143)

Retrieves the list of current actors

Retrieves the list of current actors
void **getCurrent** ()
getId (line 196)
void **getId** ()
getList (line 109)

This function returns a database array containing all objects.

This function returns a database array containing all objects.
void **getList** ()
getName (line 201)
void **getName** ()

morph (line 83)

This function can be used to turn a current object into another object. Primary Key.

This function can be used to turn a current object into another object. The object to be "morph"ed to is specified by the Primary Key.

void **morph** (*mixed* **$id**)

newId (line 120)

Return the next available ID number from the database for this object

Return the next available ID number from the database for this object

void **newId** ()

remove (line 71)

This function will delete a specified object from the database.

This function will delete a specified object from the database.

void **remove** (*mixed* **$temp**)

save (line 36)

This function saves all current information for the object, and writes the information to the database.

This function saves all current information for the object, and writes the information to the database.

void **save** ()

setCurrent (line 154)

Uses the $_GET array to set current values

Uses the $_GET array to set current values

void **setCurrent** ()

setId (line 180)

void **setId** (*mixed* **$id**)

setName (line 185)

void **setName** (*mixed* **$name**)

Documentation generated on Thu, 3 Nov 2005 21:03:35 -0700 by phpDocumentor 1.3.0RC3

Class Brain

Description

Description | Methods (details)

This class represents the learning object where the learning mechanism resides.

This class represents the learning object where the learning mechanism resides.

*Located in **/lib/Brain.php** (line **6**)*

Method Summary

Description | *Methods* (details)

Brain **Brain** ()
void **askUser** ()
void **learn** (*mixed* **$path**, *mixed* **$action**)
void **learn2** (*mixed* **$path**, *mixed* **$action**)
void **stimulus** (*mixed* **$action**)

Methods

Description | Methods (*details*)

Constructor Brain (line 8)
Brain **Brain** ()

askUser (line 213)

This function is a temporary function used to mimic

This function is a temporary function used to mimic a user response
void **askUser** ()

learn (line 32)

This function performs the "learning" and pattern analysis

This function performs the "learning" and pattern analysis
void **learn** (*mixed* **$path**, *mixed* **$action**)

learn2 (line 155)

This function performs the "learning" and pattern analysis

This function performs the "learning" and pattern analysis
void **learn2** (*mixed* **$path**, *mixed* **$action**)

stimulus (line 19)

This function is called when an action is performed

This function is called when an action is performed
void **stimulus** (*mixed* **$action**)

Documentation generated on Thu, 3 Nov 2005 21:03:35 -0700 by phpDocumentor 1.3.0RC3

Class DAB

Description

Description | Vars (details) | Methods (details)

AUTHOR: Jeremiah K. COPYRIGHT: Jeremiah K. DISCLAIMER: This code may only be used with permission from the author.

AUTHOR: Jeremiah K. Jones COPYRIGHT: Jeremiah K. Jones DISCLAIMER: This code may only be used with permission from the author. DESCRIPTION: This class is used to connect to and query a database. It requires global variables to be set in order to connect to the database. One of those fields must specify the database type. This script is compatible with MySQL, MYSQLI and MSSQL databases.

*Located in **/lib/DAB.php** (line **14**)*

Variable Summary

Description | *Vars* (details) | Methods (details)

mixed **$db**
mixed **$db_type**
mixed **$host**
mixed **$link**
mixed **$num_results**
mixed **$passwd**
mixed **$results**
mixed **$usr**

Method Summary

Description | Vars (details) | *Methods* (details)

DAB **DAB** ()
void **close** ()
void **connect** ()
void **getType** ()
void **nextRow** ()
void **numResults** ()
void **ping** ()
void **query** (*mixed* **$q**)
void **select** (*mixed* **$q**)
void **selectMultiple** (*mixed* **$q**)

Variables

Description | Vars (*details*) | Methods (details)

mixed **$db** = DB (line 22)
mixed **$db_type** = DB_TYPE (line 23)
mixed **$host** = DB_HOST (line 19)
mixed **$link** (line 26)
mixed **$num_results** (line 25)
mixed **$passwd** = DB_PASS (line 21)
mixed **$results** (line 24)
mixed **$usr** = DB_USER (line 20)

Methods

Description | Vars (details) Methods (*details*)

Constructor DAB (line 32)

Constructor

Constructor
DAB **DAB** ()

close (line 73)

Close the connection to the database

Close the connection to the database
void **close** ()

connect (line 45)

Open a connection to the database

Open a connection to the database
void **connect** ()

getType (line 239)

Return the type of database

Return the type of database
void **getType** ()
nextRow (line 203)

Retrieve the next row of information in this object's result set.

Retrieve the next row of information in this object's result set.
void **nextRow** ()
numResults (line 230)

Return the number of results in this object's result set.

Return the number of results in this object's result set.
void **numResults** ()
ping (line 249)

Tries to make a connection to the database. returns false if failed.

Tries to make a connection to the database. Returns true if success, returns false if failed.
void **ping** ()
query (line 99)

Run a query.

Run a query. This will return a database result or false if it was unsuccessful
void **query** (*mixed* **$q**)
select (line 131)

Run an SQL select query, returning only the FIRST result in the set

Run an SQL select query, returning only the FIRST result in the set
void **select** (*mixed* **$q**)
selectMultiple (line 168)

Run an SQL select statement. database results. be used.

Run an SQL select statement. The 'results' variable is loaded with the database results. In order to retrieve these, the nextRow() funciton must be used. The function returns the number of results.
void **selectMultiple** (*mixed* **$q**)

Documentation generated on Thu, 3 Nov 2005 21:03:35 -0700 by phpDocumentor 1.3.0RC3

Class Device

Description

Description | Vars (details) | Methods (details)

AUTHOR/CONTENT-OWNER: Jeremiah K. COPYRIGHT: Jeremiah K.
DISCLAIMER: This code may only be used with permission from the author
DESCRIPTION: This class represents a "Device" object. to be used with a "Device"

table in a SQL Database. contained within this class are for use with manipulating and retrieving information from the "Device" table.

AUTHOR/CONTENT-OWNER: Jeremiah K. Jones COPYRIGHT: Jeremiah K. Jones 2005 DISCLAIMER: This code may only be used with permission from the author DESCRIPTION: This class represents a "Device" object. The variables and methods are to be used with a "Device" table in a SQL Database. The variables and functions contained within this class are for use with manipulating and retrieving information from the "Device" table.

*Located in **/lib/Device.php** (line **14**)*

Variable Summary

Description | *Vars* (details) | Methods (details)

mixed **$cols**
mixed **$id**
mixed **$name**

Method Summary

Description | Vars (details) | *Methods* (details)

Device **Device** ()
void **addCurrent** (*mixed* **$actorId**, *mixed* **$timeIn**, *mixed* **$valArray**)
void **clear** ()
void **clearCurrent** ()
void **exists** (*mixed* **$id**)
void **getCount** ()
void **getCurrent** ()
void **getId** ()
void **getList** ()
void **getName** ()
void **morph** (*mixed* **$id**)
void **newId** ()
void **remove** (*mixed* **$temp**)
void **save** ()
void **setCurrent** ()
void **setId** (*mixed* **$id**)
void **setName** (*mixed* **$name**)

Variables

Description | Vars (*details*) | Methods (details)

mixed **$cols** = "id, name" (line 21)
mixed **$id** (line 19)
mixed **$name** (line 20)

Methods

Description | Vars (details) Methods (*details*)

Constructor Device (line 26)

This is the constructor

This is the constructor
Device **Device** ()

addCurrent (line 173)

Adds a current device to the table

Adds a current device to the table
void **addCurrent** (*mixed* **$actorId**, *mixed* **$timeIn**, *mixed* **$valArray**)

clear (line 58)

This function clears out the information for the current object, and deletes it from the database.

This function clears out the information for the current object, and deletes it from the database.
void **clear** ()
clearCurrent (line 131)

Clears out current Devices.

Clears out current Devices. If an id is specified, then it only clears that id
void **clearCurrent** ()
exists (line 97)

This function checks to see whether or not a specified object exists.

This function checks to see whether or not a specified object exists.
void **exists** (*mixed* **$id**)
getCount (line 47)

This class will return the total number of objects in the database

This class will return the total number of objects in the database
void **getCount** ()
getCurrent (line 143)

Retrieves the list of current devices

Retrieves the list of current devices
void **getCurrent** ()
getId (line 201)
void **getId** ()
getList (line 109)

This function returns a database array containing all objects.

This function returns a database array containing all objects.
void **getList** ()
getName (line 206)
void **getName** ()
morph (line 83)

This function can be used to turn a current object into another object. Primary Key.

This function can be used to turn a current object into another object. The object to be "morph"ed to is specified by the Primary Key.
void **morph** (*mixed* **$id**)
newId (line 120)

Return the next available ID number from the database for this object

Return the next available ID number from the database for this object

void **newId** ()
remove (line 71)

This function will delete a specified object from the database.

This function will delete a specified object from the database.
void **remove** (*mixed* **$temp**)
save (line 36)

This function saves all current information for the object, and writes the information to the database.

This function saves all current information for the object, and writes the information to the database.
void **save** ()
setCurrent (line 154)

Uses the $_GET array to set current values

Uses the $_GET array to set current values
void **setCurrent** ()
setId (line 185)
void **setId** (*mixed* **$id**)
setName (line 190)
void **setName** (*mixed* **$name**)

Documentation generated on Thu, 3 Nov 2005 21:03:36 -0700 by phpDocumentor 1.3.0RC3

Class Path

Description

Description | Vars (details) | Methods (details)

This class represents the context of a situation. the sense that it is a contextual path leading up to an event.

This class represents the context of a situation. It is called a "path" in the sense that it is a contextual path leading up to an event.

*Located in **/lib/Path.php** (line **6**)*

Variable Summary

Description | *Vars* (details) | Methods (details)

mixed **$cols**
mixed **$count**
mixed **$id**

Method Summary

Description | Vars (details) | *Methods* (details)

Path **Path** ()
void **clear** ()
void **exists** (*mixed* **$id**)
void **getCount** ()
void **getCurrActors** ()
void **getCurrDevices** ()

void **getCurrProcesses** ()
void **getId** ()
void **getList** ()
void **morph** (*mixed* **$id**)
void **newId** ()
void **remove** (*mixed* **$temp**)
void **save** ()
void **setCount** (*mixed* **$count**)
void **setId** (*mixed* **$id**)
void **snapshot** ()

Variables

Description | Vars (*details*) | Methods (details)

mixed **$cols** = "id, count" (line 10)
mixed **$count** (line 9)
mixed **$id** (line 8)

Methods

Description | Vars (details) Methods (*details*)

Constructor Path (line 12)
Path **Path** ()
clear (line 134)

This function clears out the information for the current object, and deletes it from the database.

This function clears out the information for the current object, and deletes it from the database.
void **clear** ()
exists (line 172)

This function checks to see whether or not a specified object exists.

This function checks to see whether or not a specified object exists.
void **exists** (*mixed* **$id**)
getCount (line 228)
void **getCount** ()
getCurrActors (line 42)

Returns CS list of actors

Returns CS list of actors This function returns a Comma-separated list of current actors
void **getCurrActors** ()
getCurrDevices (line 67)

Returns CS list of devices

Returns CS list of devices This function returns a Comma-separated list of current devices
void **getCurrDevices** ()
getCurrProcesses (line 98)

Returns CS list of processes

Returns CS list of processes This function returns a Comma-separated list of current processes
void **getCurrProcesses** ()
getId (line 223)
void **getId** ()
getList (line 184)

This function returns a database array containing all objects.

This function returns a database array containing all objects.
void **getList** ()
morph (line 159)

This function can be used to turn a current object into another object. Primary Key.

This function can be used to turn a current object into another object. The object to be "morph"ed to is specified by the Primary Key.
void **morph** (*mixed* **$id**)
newId (line 195)

Return the next available ID number from the database for this object

Return the next available ID number from the database for this object
void **newId** ()
remove (line 147)

This function will delete a specified object from the database.

This function will delete a specified object from the database.
void **remove** (*mixed* **$temp**)
save (line 122)

This function saves all current information for the object, and writes the information to the database.

This function saves all current information for the object, and writes the information to the database.
void **save** ()
setCount (line 212)
void **setCount** (*mixed* **$count**)
setId (line 207)
void **setId** (*mixed* **$id**)
snapshot (line 26)

Takes a "snapshot" of current environment

Takes a "snapshot" of current environment This function is used to take a snapshot of the current environment including: people interacting with the system, date/time, ext devices, activities. This function returns a string containing a "path". in standard format: A:[each actor id separated by a comma]|D:[each device id separated by a comma]|P:[each process id separated by a comma] For example: A:0,4,8:D:23,54:P:12,23,23,45,554,676
void **snapshot** ()

Documentation generated on Thu, 3 Nov 2005 21:03:36 -0700 by phpDocumentor 1.3.0RC3

Class Pattern

Description

Description | Vars (details) | Methods (details)

AUTHOR/CONTENT-OWNER: Jeremiah K. COPYRIGHT: Jeremiah K. DISCLAIMER: This code may only be used with permission from the author DESCRIPTION: This class represents a "Pattern" object. to be used with a "Pattern" table in a SQL Database. contained within this class are for use with manipulating and retrieving information from the "Pattern" table.

AUTHOR/CONTENT-OWNER: Jeremiah K. Jones COPYRIGHT: Jeremiah K. Jones 2005 DISCLAIMER: This code may only be used with permission from the author DESCRIPTION: This class represents a "Pattern" object. The variables and methods are to be used with a "Pattern" table in a SQL Database. The variables and functions contained within this class are for use with manipulating and retrieving information from the "Pattern" table.

*Located in **/lib/Pattern.php** (line **14**)*

Variable Summary

Description | *Vars* (details) | Methods (details)

mixed **$cols**
mixed **$id**
mixed **$outputId**
mixed **$weight**

Method Summary

Description | Vars (details) | *Methods* (details)

Pattern **Pattern** ()
void **clear** ()
void **execute** ()
void **exists** (*mixed* **$id**)
void **getCount** ()
void **getId** ()
void **getList** ()
void **getOutputId** ()
void **getWeight** ()
void **morph** (*mixed* **$id**)
void **newId** ()
void **remove** (*mixed* **$temp**)
void **save** ()
void **setId** (*mixed* **$id**)
void **setOutputId** (*mixed* **$outputId**)
void **setWeight** (*mixed* **$weight**)

Variables

Description | Vars (*details*) | Methods (details)

mixed **$cols** = "id, weight, outputId" (line 22)
mixed **$id** (line 19)
mixed **$outputId** (line 21)
mixed **$weight** (line 20)

Methods

Description | Vars (details) Methods (*details*)

Constructor Pattern (line 27)

This is the constructor

This is the constructor
Pattern **Pattern** ()

clear (line 59)

This function clears out the information for the current object, and deletes it from the database.

This function clears out the information for the current object, and deletes it from the database.
void **clear** ()
execute (line 133)

Executes the output for the current pattern

Executes the output for the current pattern
void **execute** ()
exists (line 100)

This function checks to see whether or not a specified object exists.

This function checks to see whether or not a specified object exists.
void **exists** (*mixed* **$id**)
getCount (line 48)

This class will return the total number of objects in the database

This class will return the total number of objects in the database
void **getCount** ()
getId (line 168)
void **getId** ()
getList (line 112)

This function returns a database array containing all objects.

This function returns a database array containing all objects.
void **getList** ()
getOutputId (line 178)
void **getOutputId** ()
getWeight (line 173)
void **getWeight** ()
morph (line 85)

This function can be used to turn a current object into another object. Primary Key.

This function can be used to turn a current object into another object. The object to be "morph"ed to is specified by the Primary Key.
void **morph** (*mixed* **$id**)
newId (line 123)

Return the next available ID number from the database for this object

Return the next available ID number from the database for this object
void **newId** ()
remove (line 73)

This function will delete a specified object from the database.

This function will delete a specified object from the database.
void **remove** (*mixed* **$temp**)
save (line 37)

This function saves all current information for the object, and writes the information to the database.

This function saves all current information for the object, and writes the information to the database.

void **save** ()

setId (line 147)
void **setId** (*mixed* **$id**)

setOutputId (line 157)
void **setOutputId** (*mixed* **$outputId**)

setWeight (line 152)
void **setWeight** (*mixed* **$weight**)

Documentation generated on Thu, 3 Nov 2005 21:03:36 -0700 by phpDocumentor 1.3.0RC3

Class Process

Description

Description | Vars (details) | Methods (details)

AUTHOR/CONTENT-OWNER: Jeremiah K. COPYRIGHT: Jeremiah K. DISCLAIMER: This code may only be used with permission from the author DESCRIPTION: This class represents a "process" object. to be used with a "process" table in a SQL Database. contained within this class are for use with manipulating and retrieving information from the "process" table.

AUTHOR/CONTENT-OWNER: Jeremiah K. Jones COPYRIGHT: Jeremiah K. Jones 2005 DISCLAIMER: This code may only be used with permission from the author DESCRIPTION: This class represents a "process" object. The variables and methods are to be used with a "process" table in a SQL Database. The variables and functions contained within this class are for use with manipulating and retrieving information from the "process" table.

*Located in **/lib/Process.php** (line **14**)*

Variable Summary

Description | *Vars* (details) | Methods (details)

mixed **$cols**
mixed **$desc**
mixed **$id**
mixed **$name**
mixed **$pName**
mixed **$runPath**

Method Summary

Description | Vars (details) | *Methods* (details)

Process **Process** ()
void **clear** ()
void **clearCurrent** ()
void **exists** (*mixed* **$id**)
void **getCount** ()
void **getCurrent** ()
void **getDesc** ()
void **getId** ()
void **getList** ()
void **getName** ()
void **getPName** ()
void **getRunPath** ()

void **kill** (*String* **$process**, *[int* **1**)
void **launchNow** (*String* **0**, *[String* **1**)
void **morph** (*mixed* **$id**)
void **newId** ()
void **processExists** (*mixed* **$pid**)
void **ps** ()
void **psIgnore** (*String* **$processName**)
void **remove** (*mixed* **$temp**)
void **save** ()
void **setDesc** (*mixed* **$desc**)
void **setId** (*mixed* **$id**)
void **setName** (*mixed* **$name**)
void **setPName** (*mixed* **$pName**)
void **setRunPath** (*mixed* **$runPath**)
void **syncProcesses** ()

Variables

Description | Vars (*details*) | Methods (details)

mixed **$cols** = "id, name, desc, pName, runPath" (line 24)
mixed **$desc** (line 21)
mixed **$id** (line 19)
mixed **$name** (line 20)
mixed **$pName** (line 22)
mixed **$runPath** (line 23)

Methods

Description | Vars (details) Methods (*details*)

Constructor Process (line 29)

This is the constructor

This is the constructor
Process **Process** ()
clear (line 61)

This function clears out the information for the current object, and deletes it from the database.

This function clears out the information for the current object, and deletes it from the database.
void **clear** ()
clearCurrent (line 140)

Clears out current Processes.

Clears out current Processes. If an id is specified, then it only clears that id
void **clearCurrent** ()
exists (line 106)

This function checks to see whether or not a specified object exists.

This function checks to see whether or not a specified object exists.
void **exists** (*mixed* **$id**)
getCount (line 50)

This class will return the total number of objects in the database

This class will return the total number of objects in the database

void **getCount** ()
getCurrent (line 152)

Retrieves the list of current devices

Retrieves the list of current devices
void **getCurrent** ()
getDesc (line 422)
void **getDesc** ()
getId (line 412)
void **getId** ()
getList (line 118)

This function returns a database array containing all objects.

This function returns a database array containing all objects.
void **getList** ()
getName (line 417)
void **getName** ()
getPName (line 427)
void **getPName** ()
getRunPath (line 432)
void **getRunPath** ()
kill (line 226)

Kills the specified process This function is used to kill a specified process running on the host machine.

Kills the specified process This function is used to kill a specified process running on the host machine.
void **kill** (*String* **$process**, *[int* **1**)

- *[int **1**: 0 if specified process is the image name 1 if PID default = 0]*
- *String **$process**: process to be killed*

launchNow (line 202)

Launches the specified application This function is used to launch an application.

Launches the specified application This function is used to launch an application.
void **launchNow** (*String* **0**, *[String* **1**)

- *String **0**: path to application to launch*
- *[String **1**: if specified, represents additional arguments to pass to the application]*

morph (line 89)

This function can be used to turn a current object into another object. Primary Key.

This function can be used to turn a current object into another object. The object to be "morph"ed to is specified by the Primary Key.

void **morph** (*mixed* **$id**)

newId (line 129)

Return the next available ID number from the database for this object

Return the next available ID number from the database for this object

void **newId** ()

processExists (line 248)

Checks to see if a given process is currently running

Checks to see if a given process is currently running

void **processExists** (*mixed* **$pid**)

ps (line 263)

Lists the current processes This function is used to list all of the current processes running on the host machine.

Lists the current processes This function is used to list all of the current processes running on the host machine.

void **ps** ()

psIgnore (line 303)

Returns true if the specified process is on the ignore list This function is used to determine whether a given process is on the ignore list.

Returns true if the specified process is on the ignore list This function is used to determine whether a given process is on the ignore list.

void **psIgnore** (*String* **$processName**)

- *String* ***$processName****: process name*

remove (line 77)

This function will delete a specified object from the database.

This function will delete a specified object from the database.

void **remove** (*mixed* **$temp**)

save (line 39)

This function saves all current information for the object, and writes the information to the database.

This function saves all current information for the object, and writes the information to the database.

void **save** ()

setDesc (line 391)

void **setDesc** (*mixed* **$desc**)

setId (line 381)

void **setId** (*mixed* **$id**)

setName (line 386)

void **setName** (*mixed* **$name**)
setPName (line 396)
void **setPName** (*mixed* **$pName**)
setRunPath (line 401)
void **setRunPath** (*mixed* **$runPath**)
syncProcesses (line 164)

Synchronizes the current process list in the database with the current OS process list.

Synchronizes the current process list in the database with the current OS process list.
void **syncProcesses** ()

Documentation generated on Thu, 3 Nov 2005 21:03:36 -0700 by phpDocumentor 1.3.0RC3

Appendix F: Source Code

/do/endProcess.php

```
<?php
	/*
	 *	This file acts as the controller for terminating a process.
	 *	The process ID is passed as a GET parameter "pid".
	 *	An additional "forward" parameter specifies where the page
	 *	redirect to once the process has been terminated.
	 */

	/*	Include the global configuration file	*/
	include_once("../etc/conf.php");

	/*	Kill the process based on the GET variable, "pid"	*/
	Process::kill($_GET['pid']);

	/*	Redirect the page based on the "forward" GET variable*/
	header("Location: ".$_GET['forward']);
?>
```

/do/launchApp.php

```
<?php
	/*
	 *	This file acts as the controller for launching an application.
	 *	Before launching the application, the "stimulus" function is
	 *	called in order to trigger the learning process based on the
	 *	action that is being called for.
	 */

	/*	Include the global configuration file	*/
	include_once("../etc/conf.php");

	/*	Create a process object based on the process ID found in the URL
	*/
	$app = new Process();
	$app->morph($_GET['processId']);

	/*	Trigger the learning mechanism	*/
	Brain::stimulus($app->id);

	/*	Launch the application	*/
	$app->launchNow();

	/*	Redirect the page based on the "forward" GET variable*/
	header("Location: ".$_GET['forward']);
?>
```

/do/systemStatus.php

```
<?php
	/*
	 *	This file acts as a simple web service for working
	 *	with the actor and device tables in the database.
	 *	Several basic functions have already been created,
	 *	but it would be good to make this more functional.
```

```
         *
         *     The desired action should be sent as a GET valud of the
         *     "action" variable.  Currently available actions include
         *     the following:
         *     clearactors: Clears out the actor table
         *     cleardevices: Clears out the devices table
         *     setactors:          Sets the actors table based on the variables
         *                               passed into the URL
         *     setdevices:         Sets the devices table based on the variables
         *                               passed into the URL
         */

        /*     Include the global confiruation file    */
        include_once("../etc/conf.php");

        /*     Retrieve the action from the GET variable "action"    */
        $action = strtolower($_GET['action']);

        /*     Perform the requested action based on the "action" specified
        */
        switch($action)
        {
               case "clearactors":
                      Actor::clearCurrent();
                      echo 'true';
                      break;

               case "cleardevices":
                      Device::clearCurrent();
                      echo 'true';
                      break;

               case "setactors":
                      Actor::clearCurrent();
                      Actor::setCurrent();
                      echo 'true';
                      break;

               case "setdevices":
                      Device::clearCurrent();
                      Device::setCurrent();
                      echo 'true';
                      break;

               default :
                      echo 'false';
                      break;
        }//end switch
?>
```

/do/triggerEvent.php

```
<?php
        /*
         *     This file acts as a simple web service for triggering
         *     the learning mechanism basd on a specified output ID.
         */

        /*     Include the global configurations */
        include_once("../etc/conf.php");

        /*     Retrieve the output ID to learn   */
        $outputID = $_GET['output'];
```

```
	/*	If the ID is not numeric, or is empty then return false and exit
	*/
	if(!is_numeric($outputID) || $outputID == '')
	{
		echo 'false';
		exit();
	}//end if

	/*	Trigger the learning mechanism	*/
	Brain::stimulus($outputID);
	echo 'true';
?>
```

/etc/Commands.php

```
<?php
	/*
	 *	This file contains all of the native commands used by
	 *	the environment.  If the system is ported to another platform,
	 *	then this file should be replaced by the equivalent commands
	 *	for that platform.
	 *
	 *	This file is automatically included by the /etc/conf.php file
	 */

	/*	Command to list the processes currently running	*/
	define("PS", "TASKLIST /S localhost /U ".USER);

	/*	Command to terminate a process based on the name	*/
	define("KILLNAME", "TASKKILL /f /t /IM");

	/*	Command to terminate a process based on the PID	*/
	define("KILLPID", "TASKKILL /f /t /PID");
?>
```

/etc/conf.php

```
<?php
	/*
	 *	This file contains all of the global configurations for the
environment.
	 *  If a certain value is likely to change based on the configuration,
	 *	then the value should first be represented in the /etc/custom.php
file so
	 *	that quick and simple modifications can be made to the commonly
changed
	 *	environment settings.
	 */

	/*	Start the session	*/
	session_start();

	/*	Include the custom configurations that are commonly edited	*/
	include_once('custom.php');

	/*	Define all necessary variables	*/
	define("USER", $localUser);		// Local System User
	define("DB_HOST",$host);		// Database Host
	define("DB_USER",$user);		// Database User
	define("DB_PASS",$passwd);		// Database Password
	define("DB",$database);			// Database name
	define("DB_TYPE", $db_type);	// Database type
```

```
	define("ROOT_URL", $url);		// Root URL
	define("ROOT_PATH", $root);			// Root Application Path
	define("LIB", ROOT_PATH."lib/");// Path to the library
	define("NUM_DEV_VALS", $num_device_values);// Number of device values in
the database table
	define("UKP", $use_known_processes);// Whether or not to ignore unknown
processes
	define("DEFAULT_TEMPLATE", $default_template);// Default UI to use

	/*	Dynamically include all php files in the library	*/
	$dir = opendir(LIB);
	while(false !== ($file = readdir($dir)))
		if(substr($file, -3) == "php")
			include_once(LIB.$file);

	/*	Additional includes */
	include_once('Commands.php');
?>
```

/etc/custom.php

```
<?php
	/*
	 *	This file contains all of the configurations for the
	 *	environment.  These values should be set to suit the needs
	 *	of the given application.
	 */

	/*	Set the database values	*/
	$host = "localhost";				//The host for the database
	$user = "contextable";				//This is the database user
	$passwd = "c0nt3xt@bl3";		//This is the database password
	$database = "ContexTable";		//This is the database name
	$db_type = "MYSQL";				//Database type - must be MYSQL,
MYSQLI or MSSQL

	/*	Set misc. variables for the learning process	*/
	$num_device_values = 3;	//Number of values in device table

	/*	Set application infrastructure information	*/
	$localUser = "Jeremiah";		// The local running user
	$root = '/cvsroot/ContexTable/www/';	//path to main site
	$url = "http://localhost/";			// Root URL
	$default_template = "default";	// The name of the default UI to use


	/*
		Configuration Variables
	*/
	/*
	 *	The "use_known_processes" directive tells the system
	 *	whether or not to include unknown processes in the current
	 *	process list.  Although there will still be some filtering done
	 *	on the list, with this directive set to "0", the process list
	 *	could grow quite long.  It is also important to note that if
	 *	unknown processes are included, then the path will include the
full
	 *	process name, as opposed to only including the known process ID.
	 */
	$use_known_processes = 1;
?>
```

/lib/Actor.php

```
<?php
	/*
		AUTHOR/CONTENT-OWNER: Jeremiah K. Jones
		COPYRIGHT: Jeremiah K. Jones 2005
		DISCLAIMER: This code may only be used with permission from
				the author

		DESCRIPTION:
		This class represents a "Actor" object.  The variables and methods
are
		to be used with a "Actor" table in a SQL Database.  The variables
and functions
		contained within this class are for use with manipulating and
retrieving
		information from the "Actor" table.
	*/

	class Actor
	{
		/*
			The variable names should mirror the columns in the
database.
		*/
		var $id;
		var $name;
		var $cols = "id, name";

		/*
			This is the constructor
		*/
		function Actor()
		{
			return true;
		}//end function Actor


		/*
			This function saves all current information for the object,
and writes
			the information to the database.
		*/
		function save()
		{
			if(!DAB::query('DELETE FROM Actor WHERE id="'.$this-
>id.'"')) return false;
			if(!DAB::query('INSERT INTO Actor ('.$this->cols.')
VALUES("'.$this->id.'", "'.$this->name.'")')) return false;
			return true;
		}//end save


		/*
			This class will return the total number of objects in the
database
		*/
		function getCount()
		{
			$temp = DAB::select("SELECT COUNT(*) FROM Actor");
			return $temp["COUNT(*)"];
		}//end getCount
```

```
		/*
			This function clears out the information for the current
			object, and deletes it from the database.
		*/
		function clear()
		{
			if(!DAB::query('DELETE FROM Actor WHERE id="'.$this-
>id.'"')) return false;

			$this->id = "";
			$this->name = "";
			return true;
		}//end clear


		/*
			This function will delete a specified object from the
database.
		*/
		function remove($temp)
		{
			if(!DAB::query('DELETE FROM Actor WHERE id="'.$temp.'"'))
return false;
			return true;
		}//end remove


		/*
			This function can be used to turn a current object into
			another object.  The object to be "morph"ed to is specified
by the
			Primary Key.
		*/
		function morph($id)
		{
			$this->id = $id;
			$row = DAB::select("SELECT * FROM Actor WHERE id=\"$this-
>id\"");
			$this->id = $row[id];
			$this->name = $row[name];

			return true;
		}//end morph


		/*
			This function checks to see whether or not a specified
object exists.
		*/
		function exists($id)
		{
			$row = DAB::select("SELECT * FROM Actor WHERE id=\"$id\"");
			if($id == $row[id])
				return true;
			return false;
		}//end exists


		/*
			This function returns a database array containing all
objects.
		*/
		function getList()
```

```
		{
			$db = new DAB();
			$db->selectMultiple("SELECT * FROM Actor ORDER BY id
DESC");
			return $db;
		}//end getList


		/*
			Return the next available ID number from the database for
this object
		*/
		function newId()
		{
			$temp = DAB::select('SELECT MAX(id) as id FROM Actor');
			return $temp['id']+1;
		}//end newId


		/*
			Clears out current Actors.  If an id is specified, then it
			only clears that id
		*/
		function clearCurrent()
		{
			if(func_num_args() > 0) $where = ' WHERE
id="'.func_get_arg(0).'"';
			else $where = '';
			DAB::query('DELETE FROM currActors'.$where);
			return true;
		}//end function


		/*
			Retrieves the list of current actors
		*/
		function getCurrent()
		{
			$db = new DAB();
			$db->selectMultiple('SELECT * FROM currActors ORDER BY
actorId');
			return $db;
		}//end function


		/*
		 *	Uses the $_GET array to set current values
		 */
		function setCurrent()
		{
			foreach($_GET as $key => $value)
			{
				if(strtolower(substr($key, 0, 3)) == "aid")
				{
					$num = substr($key, 3);
					Actor::addCurrent($value,
$_GET['atime'.$num]);
				}//end if
			}//end foreach
		}//end function


		/*
```

```
		 *	Adds a current actor to the table
		 */
		function addCurrent($actorId, $timeIn)
		{
			DAB::query('INSERT INTO currActors VALUES
("'.$actorId.'","'.$timeIn.'")');
		}//end function


		/*
			This following series of functions are used to set the
given
			variable for the current object.
		*/

		function setId($id)
		{
			$this->id = $id;
		}// end setId

		function setName($name)
		{
			$this->name = $name;
		}// end setName


		/*
			The following functions will return the current value of
the
			given variable for the current object.
		*/

		function getId()
		{
			return $this->id;
		}// end getId

		function getName()
		{
			return $this->name;
		}// end getName
	}//end class Actor
?>
```

/lib/Brain.php

```
<?
	class Brain
	{
		function Brain(){}

		/*
		 *	This is event-driven... so when an event occurs,
		 *	we take a "snapshot" of the current status (ie
		 *	actors, constraints, previous actions, etc)
		 */

		/*
		 *	This function is called when an action is performed
		 */
		function stimulus($action)
		{
			/*	Take a "snapshot" of the current environment	*/
```

```
                $thisPathID = Path::snapshot();

                /*     Analyze the snapshot and learn from it  */
                Brain::learn2($thisPathID, $action);
        }//end function stimulus


        /*
         *     This function performs the "learning" and pattern analysis
         */
        function learn($path, $action)
        {
                $lowerThresh = 3;
                $upperThresh = 6;
                $pidTimeout = 6;


                /*     Check to see if this is a pattern */
                if(Pattern::exists($path))
                {
                        /*     Retrieve the matched pattern     */
                        $pattern = new Pattern();
                        $pattern->morph($path);

                        /*     Check to see if this is a strong pattern
        */
                        if($pattern->getWeight() >= $upperThresh)
                        {
                                /*     Execute the output and retrieve the
process ID    */
                                $pid = $pattern->execute();

                                /*     Sleep for 4 seconds */
                                sleep(2);

                                /*     Watch the process to see if the user
"kills" it    */
                                $pidExists = true;
                                $startTime = time();
                                while($pidExists =
Process::processExists($pid))
                                {
                                        /*     Check the elapsed time     */
                                        if((time() - $startTime) >=
$pidTimeout)
                                                break;
                                }//end while process exists (or timeout)

                                /*     Check to see if the process was killed
        */
                                if(!$pidExists)
                                {
                                        /*     Reset the weight to the lower
threshold    */
                                        $pattern->setWeight($lowerThresh);
                                        $pattern->save();
                                }//end if the process was killed
                                else   /*     else the process was not killed
        */
                                {
                                        /*     Do nothing, just continue on
        */
                                }//end else process was not killed
```

```
                        }//end if strong pattern
                        else /*      else it is not a strong pattern  */
                                /*    Check to see if it is above the lower
thresh */
                              if($pattern->getWeight() >= $lowerThresh)
                              {
                                    /*     Prompt the user to proceed with
the output
                                           This currently calls a function
that simply returns "yes"
                                    */
                                    $response = Brain::askUser($pattern-
>getOutputId());

                                    /*     Check to see if the user wants to
proceed      */
                                    if($response == "yes")
                                    {
                                          /*     Execute the output  */
                                          $pattern->execute();

                                          /*     Strengthen the weight of
the pattern  */
                                          $pattern->setWeight($pattern-
>getWeight()+1);

                                          $pattern->save();
                                    }//end if user wants to proceed
                                    else   /*     the user does not want to
proceed      */
                                    {
                                          /*     Decrease the weight of the
pattern      */
                                          $pattern->setWeight($pattern-
>getWeight()-1);

                                          $pattern->save();
                                    }//end else the user does not want to
proceed
                              }//end if above lower thresh
                              else   /*     else is not above lower thresh
(very weak pattern) */
                              {
                                    /*     Retrieve the next expected action
      */
                                    $nextAction = $pattern->getOutputId();

                                    /*
                                          Get the real next action and
compare it...
                                          This code is still INCOMPLETE
                                    */

                                    /*     Check to see if the user's output
matches the expected output      */
                                    if($nextAction == $pattern-
>getOutput())
                                    {
                                          /*     Strengthen the weight of
the pattern  */
                                          $pattern->incrementWeight();
                                    }//end if matches expected output
                                    else   /*     else does not match output
      */
                                    {
```

```
                                                /*     Do nothing, just continue
on     */
                                                }//end else does not match output
                                        }//end else is a very weak pattern
                                }//end if is a pattern
                                else /*       else it is NOT a pattern   */
                                {
                                        /*     Check to see if this matches a path     */
                                        if(Path::exists($path))
                                        {
                                                /*     Create a new pattern       */
                                                $newPattern = new Pattern();
                                                $newPattern->setId($path);
                                                /*     Set the pattern's weight to the lower
threshold     */
                                                $newPattern->setWeight($lowerThresh);
                                                /*     Set the output to the current action

        */
                                                $newPattern->setOutputId($action);
                                                /*     Delete the path (database garbage
collection)   */
                                                $newPattern->save();
                                                Path::remove($path);
                                        }//end if matches previous path
                                        else   /*     else this does not match a path   */
                                        {
                                                /*     Save the path for future comparisons
        */
                                                $newPath = new Path();
                                                $newPath->setId($path);
                                                $newPath->save();
                                        }//end else does not match a path
                                }//end else is NOT a pattern
                        }//end function learn


                        /*
                         *     This function performs the "learning" and pattern analysis
                         */
                        function learn2($path, $action)
                        {
                                $thresh = 3;

                                /*     Check to see if this is a pattern*/
                                if(Pattern::exists($path))
                                {
                                        /*     Retrieve the matched pattern     */
                                        $pattern = new Pattern();
                                        $pattern->morph($path);

                                        /*     Execute the output and retrieve the process ID
        */
                                        $pid = $pattern->execute();
                                }//end if is a pattern
                                else /*       else it is NOT a pattern   */
                                {
                                        /*     Check to see if this matches a path     */
                                        if(Path::exists($path))
                                        {
                                                $newPath = new Path();
                                                $newPath->morph($path);
```

```
                                /*     If the path has been recognized at
least
                                       $thresh number of times, then call it a
pattern
                                */
                                if($newPath->getCount() >= $thresh)
                                {
                                       /*     Create a new pattern       */
                                       $newPattern = new Pattern();
                                       $newPattern->setId($path);
                                       /*     Set the output to the current
action */
                                       $newPattern->setOutputId($action);
                                       /*     Delete the path (database garbage
collection)   */
                                       $newPattern->save();
                                       Path::remove($path);
                                }//end if
                                else
                                {
                                       $newPath->setCount($newPath-
>getCount()+1);
                                       $newPath->save();
                                }//end else
                         }//end if matches previous path
                         else   /*     else this does not match a path   */
                         {
                                /*     Save the path for future comparisons
      */
                                $newPath = new Path();
                                $newPath->setId($path);
                                $newPath->setCount(1);
                                $newPath->save();
                         }//end else does not match a path
                  }//end else is NOT a pattern
            }//end function learn


            /*
             *     This function is a temporary function used to mimic
             *     a user response
             */
            function askUser(){return "yes";}//end function
      }//end class Brain
?>
```

/lib/DAB.php

```
<?php
      /*
            AUTHOR: Jeremiah K. Jones
            COPYRIGHT: Jeremiah K. Jones
            DISCLAIMER: This code may only be used with permission from
                        the author.

            DESCRIPTION:
            This class is used to connect to and query a database.  It
            requires global variables to be set in order to connect to the
            database.  One of those fields must specify the database type.
            This script is compatible with MySQL, MYSQLI and MSSQL databases.

            **This document should NEVER NEED TO BE ALTERED**
      */
```

```
	class DAB
	{
		/*
			Variables
		*/
		var $host = DB_HOST;
		var $usr = DB_USER;
		var $passwd = DB_PASS;
		var $db = DB;
		var $db_type = DB_TYPE;
		var $results;
		var $num_results;
		var $link;


		/*
			Constructor
		*/
		function DAB()
		{
			$this->host = DB_HOST;
			$this->usr = DB_USER;
			$this->passwd = DB_PASS;
			$this->db = DB;
			$this->db_type = DB_TYPE;
		}//end function DAB


		/*
			Open a connection to the database
		*/
		function connect()
		{
			if($this->db_type == "MYSQL")
			{
				$this->link = mysql_connect($this->host, $this->usr,
$this->passwd) or die(mysql_error());
				mysql_select_db($this->db) or die(mysql_error());
				return true;
			}//end if
			else if($this->db_type == "MYSQLI")
			{
				$this->link = mysqli_connect($this->host, $this->usr,
$this->passwd, $this->db) or die("Could not connect to the database.");
				return true;
			}//end if
			else if($this->db_type == "MSSQL")
			{
				$this->link = mssql_connect($this->host, $this->usr,
$this->passwd) or die("Couldn't connect to MSSQL database!");
				mssql_select_db($this->db) or die("Couldn't select
MSSQL database $this->db!");
				return true;
			}//end else
			else
				echo "No database selected.  Please review your
configuration.";
			return false;
		}//end connect


		/*
```

```
		Close the connection to the database
	*/
	function close()
	{
		if($this->db_type == "MYSQL")
		{
			mysql_close($this->link);
			return true;
		}//end if
		else if($this->db_type == "MYSQLI")
		{
			mysqli_close($this->link);
			return true;
		}//end if
		else if($this->db_type == "MSSQL")
		{
			mssql_close($this->link);
			return true;
		}//end else
		else
			echo "No database selected.  Please review your
configuration.";
		return false;
	}//end close


	/*
		Run a query.  This will return a database result or false
if it was unsuccessful
	*/
	function query($q)
	{
		$db = new DAB();
		$db->connect();
		if($db->getType() == "MYSQL")
		{
			$temp = mysql_query($q) or die(mysql_error());
			$db->close();
			return $temp;
		}//end if
		else if($db->getType() == "MYSQLI")
		{
			$temp = mysqli_query($db->link, $q) or
die(mysqli_error($db->link));
			$db->close();
			return $temp;
		}//end if
		else if($db->getType() == "MSSQL")
		{
			$temp = mssql_query($q) or die("Could not query MSSQL
database!");
			$db->close();
			return $temp;
		}//end else
		else
			echo "No database selected.  Please review your
configuration.";
		$db->close();
		return false;
	}//end function query


	/*
```

```
                        Run an SQL select query, returning only the FIRST result in
the set
                */
                function select($q)
                {
                        $db = new DAB();
                        $db->connect();
                        if($db->getType() == "MYSQL")
                        {
                                $temp = mysql_query($q) or die(mysql_error());
                                $temp = mysql_fetch_array($temp);
                                $db->close();
                                return $temp;
                        }//end if
                        else if($db->getType() == "MYSQLI")
                        {
                                $temp = mysqli_query($db->link, $q) or
die(mysqli_error($db->link));
                                $temp = mysqli_fetch_array($temp);
                                $db->close();
                                return $temp;
                        }//end if
                        else if($db->getType() == "MSSQL")
                        {
                                $temp = mssql_query($q) or die("Could not query MSSQL
database!");
                                $temp = mssql_fetch_array($temp);
                                $db->close();
                                return $temp;
                        }//end else
                        else
                                echo "No database selected.  Please review your
configuration.";
                        $db->close();
                        return false;
                }//end function query


                /*
                        Run an SQL select statement.  The 'results' variable is
loaded with the
                        database results.  In order to retrieve these, the
nextRow() funciton must
                        be used.  The function returns the number of results.
                */
                function selectMultiple($q)
                {
                        $db = new DAB();
                        $db->connect();
                        if($db->getType() == "MYSQL")
                        {
                                $this->results = mysql_query($q) or
die(mysql_error());
                                $db->close();
                                $this->num_results = mysql_num_rows($this->results);
                                return $this->num_results;
                        }//end if
                        if($db->getType() == "MYSQLI")
                        {
                                $this->results = mysqli_query($db->link, $q) or
die(mysqli_error($db->link));
                                $db->close();
                                $this->num_results = mysqli_num_rows($this->results);
```

```
                        return $this->num_results;
                }//end if
                else if($db->getType() == "MSSQL")
                {
                        $this->results = mssql_query($q) or die("Could not
query MSSQL database!");
                        $db->close();
                        $this->num_results = mssql_num_rows($this->results);
                        return $this->num_results;
                }//end else
                else
                        echo "No database selected.  Please review your
configuration.";
                $db->close();
                return false;
        }//end function query


        /*
                Retrieve the next row of information in this object's
result set.
        */
        function nextRow()
        {
                $row = "";
                if($this->db_type == "MYSQL")
                {
                        if($row =@ mysql_fetch_array($this->results)) return
$row;
                        else return false;
                }//end if
                if($this->db_type == "MYSQLI")
                {
                        if($row =@ mysqli_fetch_array($this->results)) return
$row;
                        else return false;
                }//end if
                else if($this->db_type == "MSSQL")
                {
                        if($row =@ mssql_fetch_array($this->results)) return
$row;
                        else return false;
                }//end else
                else
                        echo "No database selected.  Please review your
configuration.";
                return -1;
        }//end function


        /*
                Return the number of results in this object's result set.
        */
        function numResults()
        {
                return $this->num_results;
        }//end function


        /*
                Return the type of database
        */
        function getType()
```

```
		{
			return $this->db_type;
		}//end function


		/*
			Tries to make a connection to the database.  Returns true
if success,
			returns false if failed.
		*/
		function ping()
		{
			$db = new DAB();
			if(!$db->connect()) return false;
			$db->close();
			return true;
		}//end function
	}//end class DAB
?>
```

/lib/Device.php

```
<?php
	/*
		AUTHOR/CONTENT-OWNER: Jeremiah K. Jones
		COPYRIGHT: Jeremiah K. Jones 2005
		DISCLAIMER: This code may only be used with permission from
				the author

		DESCRIPTION:
		This class represents a "Device" object.  The variables and
methods are
		to be used with a "Device" table in a SQL Database.  The variables
and functions
		contained within this class are for use with manipulating and
retrieving
		information from the "Device" table.
	*/

	class Device
	{
		/*
			The variable names should mirror the columns in the
database.
		*/
		var $id;
		var $name;
		var $cols = "id, name";

		/*
			This is the constructor
		*/
		function Device()
		{
			return true;
		}//end function Device


		/*
			This function saves all current information for the object,
and writes
			the information to the database.
		*/
```

```
		function save()
		{
			if(!DAB::query('DELETE FROM Device WHERE id="'.$this-
>id.'"')) return false;
			if(!DAB::query('INSERT INTO Device ('.$this->cols.')
VALUES("'.$this->id.'", "'.$this->name.'")')) return false;
			return true;
		}//end save


		/*
			This class will return the total number of objects in the
database
		*/
		function getCount()
		{
			$temp = DAB::select("SELECT COUNT(*) FROM Device");
			return $temp["COUNT(*)"];
		}//end getCount


		/*
			This function clears out the information for the current
			object, and deletes it from the database.
		*/
		function clear()
		{
			if(!DAB::query('DELETE FROM Device WHERE id="'.$this-
>id.'"')) return false;

			$this->id = "";
			$this->name = "";
			return true;
		}//end clear


		/*
			This function will delete a specified object from the
database.
		*/
		function remove($temp)
		{
			if(!DAB::query('DELETE FROM Device WHERE id="'.$temp.'"'))
return false;
			return true;
		}//end remove


		/*
			This function can be used to turn a current object into
			another object.  The object to be "morph"ed to is specified
by the
			Primary Key.
		*/
		function morph($id)
		{
			$this->id = $id;
			$row = DAB::select("SELECT * FROM Device WHERE id=\"$this-
>id\"");
			$this->id = $row[id];
			$this->name = $row[name];

			return true;
```

```
		}//end morph


		/*
			This function checks to see whether or not a specified
object exists.
		*/
		function exists($id)
		{
			$row = DAB::select("SELECT * FROM Device WHERE
id=\"$id\"");
			if($id == $row[id])
				return true;
			return false;
		}//end exists


		/*
			This function returns a database array containing all
objects.
		*/
		function getList()
		{
			$db = new DAB();
			$db->selectMultiple("SELECT * FROM Device ORDER BY id
DESC");
			return $db;
		}//end getList


		/*
			Return the next available ID number from the database for
this object
		*/
		function newId()
		{
			$temp = DAB::select('SELECT MAX(id) as id FROM Device');
			return $temp['id']+1;
		}//end newId


		/*
			Clears out current Devices.  If an id is specified, then it
			only clears that id
		*/
		function clearCurrent()
		{
			if(func_num_args() > 0) $where = ' WHERE
id="'.func_get_arg(0).'"';
			else $where = '';
			DAB::query('DELETE FROM currDevices'.$where);
			return true;
		}//end function


		/*
			Retrieves the list of current devices
		*/
		function getCurrent()
		{
			$db = new DAB();
			$db->selectMultiple('SELECT * FROM currDevices ORDER BY
deviceId');
```

```
		return $db;
	}//end function


	/*
	 *	Uses the $_GET array to set current values
	 */
	function setCurrent()
	{
		foreach($_GET as $key => $value)
		{
			if(strtolower(substr($key, 0, 3)) == "did")
			{
				$num = substr($key, 3);
				$valArray = '';
				for($i=0; $i < NUM_DEV_VALS; $i++)
					$valArray[$i] = $_GET['dval'.$num.$i];
				Device::addCurrent($value,
$_GET['dtime'.$num], $valArray);
			}//end if
		}//end foreach
	}//end function


	/*
	 *	Adds a current device to the table
	 */
	function addCurrent($actorId, $timeIn, $valArray)
	{
		$valArray = implode($valArray, '","');
		DAB::query('INSERT INTO currDevices VALUES
("'.$actorId.'","'.$timeIn.'", "'.$valArray.'")');
	}//end function


	/*
		This following series of functions are used to set the
given
		variable for the current object.
	*/

	function setId($id)
	{
		$this->id = $id;
	}// end setId

	function setName($name)
	{
		$this->name = $name;
	}// end setName


	/*
		The following functions will return the current value of
the
		given variable for the current object.
	*/

	function getId()
	{
		return $this->id;
	}// end getId
```

```
		function getName()
		{
			return $this->name;
		}// end getName
	}//end class Device
?>
```

/lib/Path.php

```
<?
class Path
{
		var $id;
		var $count;
		var $cols = "id, count";
		/*	Constructor	*/
		function Path(){}


		/**
		* Takes a "snapshot" of current environment
		*
		* This function is used to take a snapshot of the current
environment
		* including: people interacting with the system, date/time, ext
devices,
		* activities.  This function returns a string containing a "path".
in
		* standard format:
		*			A:[each actor id separated by a comma]|D:[each
device id separated by a comma]|P:[each process id separated by a comma]
		*		For example:
		*			A:0,4,8:D:23,54:P:12,23,23,45,554,676
		*/
		function snapshot()
		{
			$actorList = Path::getCurrActors();
			$deviceList = Path::getCurrDevices();
			$processList = Path::getCurrProcesses();

			return
'A:'.$actorList.'|D:'.$deviceList.'|P:'.$processList;
		}//end function



		/**
		* Returns CS list of actors
		*
		* This function returns a Comma-separated list of current actors
		*/
		function getCurrActors()
		{
			$retArray = '';
			$actors = Actor::getCurrent();
			if($actors->numResults() > 0)
			{
				$i=0;
				while($row = $actors->nextRow())
				{
					$retArray[$i] = $row['actorId'];
					$i++;
				}//end while
```

```
                        return implode(",",$retArray);
                }//end if
                else
                        return false;
        }//end function



        /**
        * Returns CS list of devices
        *
        * This function returns a Comma-separated list of current devices
        */
        function getCurrDevices()
        {
                $retArray = '';
                $devices = Device::getCurrent();
                if($devices->numResults() > 0)
                {
                        $i=0;
                        while($row = $devices->nextRow())
                        {
                                $retArray[$i] = $row['deviceId'];
                                $valArray = '';
                                for($j=0; $j < NUM_DEV_VALS; $j++)
                                        $valArray[$j] = $row["val".($j+1)];
                                $valArray = array_filter($valArray);
                                if(count($valArray)>0)
                                        $retArray[$i] .= "->".implode("-
>",$valArray);
                                $i++;
                        }//end while
                        return implode(",",$retArray);
                }//end if
                else
                        return false;
        }//end function



        /**
        * Returns CS list of processes
        *
        * This function returns a Comma-separated list of current
processes
        */
        function getCurrProcesses()
        {
                Process::syncProcesses();
                $retArray = '';
                $processes = Process::getCurrent();
                if($processes->numResults() > 0)
                {
                        $i=0;
                        while($row = $processes->nextRow())
                        {
                                $retArray[$i] = $row['processId'];
                                $i++;
                        }//end while
                        return implode(",",$retArray);
                }//end if
                else
                        return false;
```

```
		}//end function


		/*
			This function saves all current information for the object,
and writes
			the information to the database.
		*/
		function save()
		{
			if(!DAB::query('DELETE FROM Path WHERE id="'.$this-
>id.'"')) return false;
			if(!DAB::query('INSERT INTO Path ('.$this->cols.')
VALUES("'.$this->id.'", "'.$this->count.'")')) return false;
			return true;
		}//end save


		/*
			This function clears out the information for the current
			object, and deletes it from the database.
		*/
		function clear()
		{
			if(!DAB::query('DELETE FROM Path WHERE id="'.$this-
>id.'"')) return false;

			$this->id = "";
			$this->count = "";
			return true;
		}//end clear


		/*
			This function will delete a specified object from the
database.
		*/
		function remove($temp)
		{
			if(!DAB::query('DELETE FROM Path WHERE id="'.$temp.'"'))
return false;
			return true;
		}//end remove


		/*
			This function can be used to turn a current object into
			another object.  The object to be "morph"ed to is specified
by the
			Primary Key.
		*/
		function morph($id)
		{
			$this->id = $id;
			$row = DAB::select("SELECT * FROM Path WHERE id=\"".$this-
>id."\"");
			$this->count = $row['count'];

			return true;
		}//end morph


		/*
```

```
                This function checks to see whether or not a specified
object exists.
        */
        function exists($id)
        {
                $row = DAB::select("SELECT COUNT(*) AS count FROM Path
WHERE id=\"$id\"");
                if($row['count'] >= 1)
                        return true;
                return false;
        }//end exists


        /*
                This function returns a database array containing all
objects.
        */
        function getList()
        {
                $db = new DAB();
                $db->selectMultiple("SELECT * FROM Path ORDER BY id DESC");
                return $db;
        }//end getList


        /*
                Return the next available ID number from the database for
this object
        */
        function newId()
        {
                $temp = DAB::select('SELECT MAX(id) as id FROM Path');
                return $temp['id']+1;
        }//end newId


        /*
                This following series of functions are used to set the
given
                variable for the current object.
        */

        function setId($id)
        {
                $this->id = $id;
        }// end setId

        function setCount($count)
        {
                $this->count = $count;
        }// end setCount


        /*
                The following functions will return the current value of
the
                given variable for the current object.
        */

        function getId()
        {
                return $this->id;
        }// end getId
```

```
		function getCount()
		{
			return $this->count;
		}// end getCount
}//end class Path
?>
```

/lib/Pattern.php

```
<?php
	/*
		AUTHOR/CONTENT-OWNER: Jeremiah K. Jones
		COPYRIGHT: Jeremiah K. Jones 2005
		DISCLAIMER: This code may only be used with permission from
					the author

		DESCRIPTION:
		This class represents a "Pattern" object.  The variables and
methods are
		to be used with a "Pattern" table in a SQL Database.  The
variables and functions
		contained within this class are for use with manipulating and
retrieving
		information from the "Pattern" table.
	*/

	class Pattern
	{
		/*
			The variable names should mirror the columns in the
database.
		*/
		var $id;
		var $weight;
		var $outputId;
		var $cols = "id, weight, outputId";

		/*
			This is the constructor
		*/
		function Pattern()
		{
			return true;
		}//end function Pattern


		/*
			This function saves all current information for the object,
and writes
			the information to the database.
		*/
		function save()
		{
			if(!DAB::query('DELETE FROM Pattern WHERE id="'.$this-
>id.'"')) return false;
			if(!DAB::query('INSERT INTO Pattern ('.$this->cols.')
VALUES("'.$this->id.'", "'.$this->weight.'", "'.$this->outputId.'")')) return
false;
			return true;
		}//end save
```

```
		/*
			This class will return the total number of objects in the
database
		*/
		function getCount()
		{
			$temp = DAB::select("SELECT COUNT(*) FROM Pattern");
			return $temp["COUNT(*)"];
		}//end getCount


		/*
			This function clears out the information for the current
			object, and deletes it from the database.
		*/
		function clear()
		{
			if(!DAB::query('DELETE FROM Pattern WHERE id="'.$this-
>id.'"')) return false;

			$this->id = "";
			$this->weight = "";
			$this->outputId = "";
			return true;
		}//end clear


		/*
			This function will delete a specified object from the
database.
		*/
		function remove($temp)
		{
			if(!DAB::query('DELETE FROM Pattern WHERE id="'.$temp.'"'))
return false;
			return true;
		}//end remove


		/*
			This function can be used to turn a current object into
			another object.  The object to be "morph"ed to is specified
by the
			Primary Key.
		*/
		function morph($id)
		{
			$this->id = $id;
			$row = DAB::select("SELECT * FROM Pattern WHERE id=\"$this-
>id\"");
			$this->id = $row[id];
			$this->weight = $row[weight];
			$this->outputId = $row[outputId];

			return true;
		}//end morph


		/*
			This function checks to see whether or not a specified
object exists.
		*/
		function exists($id)
```

```
		{
			$row = DAB::select("SELECT * FROM Pattern WHERE
id=\"$id\"");
			if($id == $row[id])
				return true;
			return false;
		}//end exists


		/*
			This function returns a database array containing all
objects.
		*/
		function getList()
		{
			$db = new DAB();
			$db->selectMultiple("SELECT * FROM Pattern ORDER BY id
DESC");
			return $db;
		}//end getList


		/*
		 *	Return the next available ID number from the database for
this object
		 */
		function newId()
		{
			$temp = DAB::select('SELECT MAX(id) as id FROM Pattern');
			return $temp['id']+1;
		}//end newId


		/*
		 *	Executes the output for the current pattern
		 */
		function execute()
		{
			$process = new Process();
			$process->morph($this->outputId);
			$process->launchNow();
			return $this->outputId;
		}//end execute


		/*
			This following series of functions are used to set the
given
			variable for the current object.
		*/

		function setId($id)
		{
			$this->id = $id;
		}// end setId

		function setWeight($weight)
		{
			$this->weight = $weight;
		}// end setWeight

		function setOutputId($outputId)
		{
```

```
                $this->outputId = $outputId;
        }// end setOutputId


        /*
                The following functions will return the current value of
the
                given variable for the current object.
        */

        function getId()
        {
                return $this->id;
        }// end getId

        function getWeight()
        {
                return $this->weight;
        }// end getWeight

        function getOutputId()
        {
                return $this->outputId;
        }// end getOutputId
}//end class Pattern
?>
```

/lib/Process.php

```
<?php
        /*
                AUTHOR/CONTENT-OWNER: Jeremiah K. Jones
                COPYRIGHT: Jeremiah K. Jones 2005
                DISCLAIMER: This code may only be used with permission from
                                the author

                DESCRIPTION:
                This class represents a "process" object.  The variables and
methods are
                to be used with a "process" table in a SQL Database.  The
variables and functions
                contained within this class are for use with manipulating and
retrieving
                information from the "process" table.
        */

        class Process
        {
                /*
                        The variable names should mirror the columns in the
database.
                */
                var $id;
                var $name;
                var $desc;
                var $pName;
                var $runPath;
                var $cols = "id, name, desc, pName, runPath";

                /*
                        This is the constructor
                */
                function Process()
```

```
		{
			return true;
		}//end function process


		/*
			This function saves all current information for the object,
and writes
			the information to the database.
		*/
		function save()
		{
			if(!DAB::query('DELETE FROM process WHERE id="'.$this-
>id.'"')) return false;
			if(!DAB::query('INSERT INTO process ('.$this->cols.')
VALUES("'.$this->id.'", "'.$this->name.'", "'.$this->desc.'", "'.$this-
>pName.'", "'.$this->runPath.'")')) return false;
			return true;
		}//end save


		/*
			This class will return the total number of objects in the
database
		*/
		function getCount()
		{
			$temp = DAB::select("SELECT COUNT(*) FROM process");
			return $temp["COUNT(*)"];
		}//end getCount


		/*
			This function clears out the information for the current
			object, and deletes it from the database.
		*/
		function clear()
		{
			if(!DAB::query('DELETE FROM process WHERE id="'.$this-
>id.'"')) return false;

			$this->id = "";
			$this->name = "";
			$this->desc = "";
			$this->pName = "";
			$this->runPath = "";
			return true;
		}//end clear


		/*
			This function will delete a specified object from the
database.
		*/
		function remove($temp)
		{
			if(!DAB::query('DELETE FROM process WHERE id="'.$temp.'"'))
return false;
			return true;
		}//end remove


		/*
```

```
                        This function can be used to turn a current object into
                        another object.  The object to be "morph"ed to is specified
by the
                        Primary Key.
                */
                function morph($id)
                {
                        $this->id = $id;
                        $row = DAB::select("SELECT * FROM process WHERE id=\"$this-
>id\"");
                        $this->id = $row[id];
                        $this->name = $row[name];
                        $this->desc = $row[desc];
                        $this->pName = $row[pName];
                        $this->runPath = $row[runPath];

                        return true;
                }//end morph


                /*
                        This function checks to see whether or not a specified
object exists.
                */
                function exists($id)
                {
                        $row = DAB::select("SELECT * FROM process WHERE
id=\"$id\"");
                        if($id == $row[id])
                                return true;
                        return false;
                }//end exists


                /*
                        This function returns a database array containing all
objects.
                */
                function getList()
                {
                        $db = new DAB();
                        $db->selectMultiple("SELECT * FROM process ORDER BY id
DESC");
                        return $db;
                }//end getList


                /*
                        Return the next available ID number from the database for
this object
                */
                function newId()
                {
                        $temp = DAB::select('SELECT MAX(id) as id FROM process');
                        return $temp['id']+1;
                }//end newId


                /*
                        Clears out current Processes.  If an id is specified, then
it
                        only clears that id
                */
```

```
		function clearCurrent()
		{
			if(func_num_args() > 0) $where = ' WHERE
id="'.func_get_arg(0).'"';
			else $where = '';
			DAB::query('DELETE FROM currProcesses'.$where);
			return true;
		}//end function


		/*
			Retrieves the list of current devices
		*/
		function getCurrent()
		{
			$db = new DAB();
			$db->selectMultiple('SELECT * FROM currProcesses ORDER BY
processId');
			return $db;
		}//end function


		/**
		* Synchronizes the current process list in the database
		* with the current OS process list.
		*/
		function syncProcesses()
		{
			Process::clearCurrent();
			$psArray = Process::ps();
			if(UKP != 1)
			{
				sort($psArray);
				foreach($psArray as $ps)
					DAB::query('INSERT INTO
currProcesses(processId) VALUES ("'.$ps.'")');
				return true;
			}//end if

			$knownPS = Process::getList();
			if($knownPS->numResults() > 0)
			{
				while($row = $knownPS->nextRow())
				{
					foreach($psArray as $ps)
					{
						if(trim(strtolower($ps)) ==
trim(strtolower($row['pName'])))
						{
							DAB::query('INSERT INTO
currProcesses(processId) VALUES ("'.$row['id'].'")');
						}//end if
					}//end foreach
				}//end while
			}//end if
			return false;
		}//end function


		/**
		* Launches the specified application
		*
		* This function is used to launch an application.
```

```
		*
		* @param	String path to application to launch
		* @param	[String if specified, represents additional arguments
to pass to the application]
		*/
		function launchNow()
		{
			/*	Determine whether or not there are arguments to pass
	*/
			if(func_num_args() > 1)
				$args = " ".func_get_arg(1);
			else $args = "";

			/*	Launch the application	*/
			echo '
			<script language="JavaScript">
				obj = new ActiveXObject("LaunchinIE.Launch");
				obj.LaunchApplication(\''.$this->runPath.$args.'\');
			</script>';
			ob_flush();
			flush();
		}//end function



		/**
		* Kills the specified process
		*
		* This function is used to kill a specified process
		* running on the host machine.
		*
		* @param	String process to be killed
		* @param	[int 0 if specified process is the image name 1 if
PID default = 0]
		*/
		function kill($process)
		{
			/*	Determine whether or not the specified process is a
PID	*/
			if(func_num_args() > 1 && func_get_arg(1) == 1)
				$isPID = true;
			else $isPID = false;

			/*	Kill the process	*/
			if($isPID)
				$ret = exec(KILLPID." $process", $tmp, $success);
			else
				$ret = exec(KILLNAME." $process", $tmp, $success);

			/*	Return true if success false if failed	*/
			if($success == 0) return true;
			return false;
		}//end function


		/**
		* Checks to see if a given process is currently running
		*/
		function processExists($pid)
		{
			$psArray = Process::ps();
			foreach($psArray as $ps)
```

```
                        if(trim(strtolower($pid)) == trim(strtolower($ps)))
return true;
                return false;
        }//end processExists


        /**
        * Lists the current processes
        *
        * This function is used to list all of the current processes
        * running on the host machine.
        */
        function ps()
        {
                $psTree = '';

                /*      This command builds an array containing all processes
      */
                exec(PS, $tmp);

                $i=0;

                /*      Filter through each processes, extracing the .exe
name   */
                foreach($tmp as $value)
                {
                        if(preg_match('/(.*\.exe)/i', $value, $matches))
                        {
                                /*      Check to see if this proces is on the
"ignore" list*/
                                if(!Process::psIgnore($matches[0]))
                                {
                                        $psTree[$i] = $matches[0];
                                        $i++;
                                }//end if
                        }//end if
                }//end foreach

                /*      Sort the array      */
                sort($psTree);

                /*      Return the process array   */
                return $psTree;
        }//end function



        /**
        * Returns true if the specified process is on the ignore list
        *
        * This function is used to determine whether a given process
        * is on the ignore list.
        *
        * @param String process name
        */
        function psIgnore($processName)
        {
                /*      First check if it is on the "good" list
                        This helps speed up the search since this list is
smaller      */
                $gArray = array(
                                        "iexplore.exe",
                                        "aim.exe",
```

```
                    "dreamweaver.exe",
                    "msmoney.exe"
                    );

foreach($gArray as $value)
      if(strtolower($value) == strtolower($processName))
            return false;

/*    Now see if it is on the "ignore" list   */
$iArray = array(
                    "agrsmmsg.exe",
                    "alcxmntr.exe",
                    "mdm.exe",
                    "nprotect.exe",
                    "wisptis.exe",
                    "apache.exe",
                    "apachemonitor.exe",
                    "defwatch.exe",
                    "hpzipm12.exe",
                    "incd.exe",
                    "rtvscan.exe",
                    "vptray.exe",
                    "viewmgr.exe",
                    "alg.exe",
                    "cidaemon.exe",
                    "cisvc.exe",
                    "csrss.exe",
                    "ctfmon.exe",
                    "dllhost.exe",
                    "ehrecvr.exe",
                    "ehsched.exe",
                    "ehmsas.exe",
                    "ehtray.exe",
                    "explorer.exe",
                    "hphmon06.exe",
                    "hpsysdrv.exe",
                    "ipodservice.exe",
                    "ituneshelper.exe",
                    "incdsrv.exe",
                    "jusched.exe",
                    "kbd.exe",
                    "lsass.exe",
                    "msimn.exe",
                    "msmsgs.exe",
                    "ntvdm.exe",
                    "nvsvc32.exe",
                    "realsched.exe",
                    "rundll32.exe",
                    "services.exe",
                    "smss.exe",
                    "spoolsv.exe",
                    "svchost.exe",
                    "tasklist.exe",
                    "winlogon.exe",
                    "winmysqladmin.exe",
                    "wmiprvse.exe"
                    );

foreach($iArray as $value)
      if(strtolower($value) == strtolower($processName))
            return true;

return false;
```

```
	}//end function


	/*
		This following series of functions are used to set the
given
		variable for the current object.
	*/

	function setId($id)
	{
		$this->id = $id;
	}// end setId

	function setName($name)
	{
		$this->name = $name;
	}// end setName

	function setDesc($desc)
	{
		$this->desc = $desc;
	}// end setDesc

	function setPName($pName)
	{
		$this->pName = $pName;
	}// end setPName

	function setRunPath($runPath)
	{
		$this->runPath = $runPath;
	}// end setRunPath


	/*
		The following functions will return the current value of
the
		given variable for the current object.
	*/

	function getId()
	{
		return $this->id;
	}// end getId

	function getName()
	{
		return $this->name;
	}// end getName

	function getDesc()
	{
		return $this->desc;
	}// end getDesc

	function getPName()
	{
		return $this->pName;
	}// end getPName

	function getRunPath()
	{
```

```
                return $this->runPath;
        }// end getRunPath
    }//end class Process
?>
```

/applications.php

```
<?php
    /*
     *    This is a sample file to show how an application
     *    Can be launched from the browser using the environment
     */

    /*    Include the global configuration file   */
    include_once("etc/conf.php");

    /*    Retrieve all of the applications  */
    $pList = Process::getList();
?>
<html>
<head />
<body>
<script>
function setFrameLoc(loc)
{
    frames['launchApp'].location.href=loc;
}//end function
</script>
<? if($pList->numResults() > 0 ) :?>
    <table cellpadding="0" cellspacing="0" border="0" width="100%">
        <? while($row = $pList->nextRow()) :?>
        <tr valign="top">
            <td><p><a
href="javascript:setFrameLoc('do/launchApp.php?processId=<?=$row['id']?>');">La
unch <?=$row['name']?></a></p></td>
        </tr>
        <? endwhile ?>
    </table>

    <iframe border=0 src="" name="launchApp" id="launchApp" width=1 height=1
style="margin: 0px"></iframe>
<? else :?>

<? endif ?>
</body>
</html>
```

/index.php

```
<?php
    /*
     *    This is the main page that is used to drive the site.
     *    The UI is loaded based on the parameters provided in the URL
     */

    /*    Include the global configurations */
    include_once("etc/conf.php");

    /*    If no template is specified, load in the default     */
    if(!$template = $_GET['t']) $template = DEFAULT_TEMPLATE;

    /*    If no page was specified, load the index page  */
    if(!$page = $_GET['p']) $page = "index.php";
```

```
        /*     Load in the proper template and page    */
        include("ui/$template/$page");
?>


/terminate.php
<?php
        /*
         *     This is a sample file for how to terminate
         *     a process from the browser
         */

        /*     Include the global configurations */
        include_once("etc/conf.php");

        /*     Retrieve all of the running processes   */
        $processes = Process::ps();
?>
<? foreach($processes as $process) :?>
        <p>
                <a href="do/endProcess.php?pid=<?=$process?>">Terminate
Process:  <?=$process?></a>
        </p>
<? endforeach ?>
```