Developer Guide

VERSION 1.0

TheraTouch

This manual outlines the development process and the standards for the TheraTouch project. The goal for this document is to aid future developers in the creation of activities, modification of the TheraTouch framework, and the TheraLink web application.

Texas Christian University

© Computer Science Department

9 May 2012

Revision Sign-off

By signing the following, the team member asserts that he/she has read the entire document and has, to the best of his or her knowledge found the information contained herein to be accurate, relevant, and free of typographical error.

NAME	SIGNATURE	DATE
SCOTT BOYKIN		
ADAM BURT		
CRISTINA CLINE		
JOHN FARRIS		
JEFF GETTEL		
ANDREW HUGHES		
KEVIN LAWHON		
MASON MCGLOTHLIN		
NIPUNA PERERA		
MANDY SEREMET		
JOEY WILKINSON		

Revision History

VERSION	DATE	CHANGES
Version 1.0	In progress	Still in planning stage

Contents

1. Introduction

TheraTouch is a system designed for therapeutic rehabilitation. It allows for custom sessions for each user and for each activity within a session to have custom options. In order to allow for these customizations, we had to create three parts that work together to make our system, *TheraTouch*. We have: the *TheraTouch* Framework, which is running on the Microsoft Surface, the *TheraLink* web application, and the TheraDB, where all information regarding users' sessions are stored.

1.1 Purpose

The purpose of this document is to help developers install TheraTouch onto their system or how to make and add new activities, or be able to make changes to the system, should the need arise. The three separate parts of TheraTouch will be outlined and examined for the developer's benefit.

2. Installation

On the project disk, there is a folder named *TheraTouch* and file named 'TheraLink.zip'. The *TheraTouch* folder contains the framework and all of the activities already packaged inside of it. The 'TheraLink.zip' folder contains the *TheraLink* web application and the database scripts used to create the TheraDB and AccountDB databases. It also contains some example scripts to show how to add activities and activity options to the database. See section 4.4 for SQL examples.

2.1.1 Create TheraTouch Database

1. First, you will want to modify the variables for each script according to the environment at where *TheraTouch* is being deployed. The variables user, password and server need to be configured correctly in 'CreateDB.bat', 'PopulateDBsample.bat', and 'DeleteDB.bat' at the top of the file.

2. Then execute the following two lines, substituting a database name of your choosing instead of "TheraLink". The machine these scripts are being run from will need the *sqlcmd* program installed (alternatively, run the scripts directly on the server).

```
CreateDB.bat TheraLink

PopulateDBsample.bat TheraLink
```

2.1.2 Create Account Database for TheraLink

The next step is to place the website code on a development machine, and open the site in Visual Studio 2010. Before actually running the project, you must create an account database (which is separate from the TheraLink database) and it will need to be configured with at least one "Manager" user.

1. On a any computer connected to the network, go to where the .NET 4.0 framework is installed (usually C:\Windows\Microsoft.NET\Framework64\v4.0.30319\) and run aspnet_regsql.exe

2. Once the wizard opens, click "Next".

3. Make sure "Configure SQL Server for application services" is selected and click "Next".

4. Enter in the SQL server address in the "Server:" field. Enter any credential information required for your SQL server. Type "AccountDB" (The database shouldn't exist yet) in the "Database:" field. Click Next.

5. A summary of the settings you have chosen will be shown on the next screen. Click "Next" if everything looks right.

6. The AccountDB should now be set up with all the proper ASP.NET membership tables. Click Finish to exit the wizard.

7. Modify the connection strings in the web.config file for the site to match the correct database server, database name, and credentials in your environment.

8. Next, the AccountDB needs to be configured. We used the ASP.NET configuration utility. In the menu bar in VS2010, click Website > ASP.NET Configuration. This should bring up the administration utility.

9. Go to Security, and then click Create Roles. We need two roles: "Clinician" and "Manager".

10. Once that is done, return to the Security tab and create a user, and assign the Manager role to this user. You can then exit the configuration utility.

2.1.3 Install/Start Web Server

Next, the site can be tested from VS2010 by running the project, or installed onto the web server. We ran it in a virtual application (.NET 4.0 must be enabled). The user that was created with the ASP.NET configuration utility will be the initial login credentials. From this point, everything should be fully operational. To create additional administrative users, the built in Admin page on the site can be used.

2.1.4 Install the TheraTouch Framework

First, find the folder called ‘TheraTouch’ located on the project disk, and copy it into the Program Files. Next, using the ‘TheraTouchV1.xml’ file located inside the ‘TheraTouch’ folder, follow the instructions provided by Microsoft on deploying an activity to the Microsoft Surface (Appendix). To function properly, the Surface needs to be in single application mode. Follow the guide provide by Microsoft to accomplish this (Appendix). You will need to edit an additional register not specified by the guide. Set the *App2ndInstanceNofification* in *ModeProfiles\Default\Shell* to 0. Finally, edit the ‘ServerData.txt file’, located in the ‘TheraTouch’ folder, with the appropriate database information.

3. Development Getting Started

In this section, we will discuss the initial steps needed to be taken in order to develop for *TheraTouch*.

3.1 TheraDB

TheraDB is the database used to store all the session and activity information and activity results, as well as the user information. To develop for *TheraDB*, it is necessary to have a developer's environment for Microsoft SQL Server 2008 R2 such as SQL Server Management Studio, as well as a server than can host *TheraDB*. *TheraDB* can be hosted on the same server as the *TheraLink* web application.

3.2 TheraLink

TheraLink is the clinician web application that is used to add and manage users, create sessions for those users, and view the results of the users' activities. To develop for *TheraLink*, Microsoft Visual Studio 2010 or Microsoft Visual Web Developer 2010 is necessary.

3.3 TheraTouch

TheraTouch is the framework that runs on the Microsoft Surface unit. Before starting development for *TheraTouch*, it is necessary to have fulfilled the following requirements in the following order:

1. A 32-bit Windows Vista or Windows 7 operating system
2. Have Visual Studio 2008 installed
3. XNA Framework Redistributable 2.0 installed
4. XNA 3.0 Game Studio
5. Surface SDK 1.0 SP1.

Note Both XNA Framework Redistributable 2.0 and XNA 3.0 Game Studio need to be installed before installing Surface SDK 1.0 SP1, it requires XNA Framework Redistributable 2.0. If XNA 3.0 Game Studio is not installed, Visual Studio will not contain the required templates needed to develop activities.

The Microsoft Surface has two modes, User Mode and Admin Mode. User Mode is used to run *TheraTouch* application, while Admin Mode lets you interact with Windows Vista just like a normal computer. Admin Mode requires a mouse and keyboard. To deploy new activities, the Surface must be in Admin Mode. If it is currently in User Mode, plug in a mouse and keyboard, and then press Ctrl-Alt-Delete. This will allow you to log out of User Mode. Once at the log in screen, you can log into the Admin account. To enter User Mode from Admin Mode, execute the application located on the desktop call, Enter User Mode.

4. TheraDB Development

4.1 Introduction

TheraDB is the database used to store all the data collected during a session. It was designed to be flexible enough to easily accommodate changes such as: adding a new activity to the framework, storing any results that are collected in these activities, and creating a custom sessions for any user.

4.2 Tables

4.2.1 UserInfo

This table is used to store users' information that is in accordance with HIPAA. It also stores information that pertains to the *TheraTouch* system such as: a user's current session, and their TheraID, a sequentially generated integer which is used to log them into their session. The primary key is the user's *TheraID*

4.2.2 Activity

This table stores all the activities that are in the framework so that the web application is able to view them and then add them to a session, and the framework running on the Surface will be able load the games. Each activity has a short name that is used as the internal identifier by TheraTouch and a long name that is used in the various user interfaces. The primary key is the *ActivityID*.

4.2.3 ActivityOptionList

This table stores all of the options that each activity can have. The options in this table are not associated with any session. *ActivityOptionName* specifies the name of the option; this will be displayed to staff users in TheraLink. *ActivityOptionType* indicates the type of this option. Currently, the only option type is "Enumerated" and this field is not actually used. However, it exists for future development to possibly support clinician-entered custom options. DefaultValue indicates what the default value is for the option. This table is related to the Activity table through the *ActivityID* field. The primary keys are *ActivityID* and *ActivityOptionID*.

4.2.4 ActivityOptionValuesEnumerated

This table stores all of the possible values for each option that an activity can have. The values in this table are not associated with any session. This table is related to the ActivityOptionList table through the *ActivityID* and *ActivityOptionID* fields. *ActivityID*, *ActivityOptionID*, and *ActivityOptionEnumID* are the primary keys.

4.2.5 SessionOptions

This table stores all of the sessions for each user, and contains multiple flags to specify different information for the *TheraTouch* framework. The *TheraID* column specifies which user the session belongs to. *Open_Closed_Status* indicates whether the session is currently available to play on the Microsoft Surface or not. Activities can be closed by *TheraTouch* after all activities in the session have been completed. Or a staff user can mark the session as closed from

TheraLink before all activities have been completed if they wish. *SessionInProgress* status is a flag that indicates if the session is currently open in *TheraTouch*. A session is in progress when open in *TheraTouch*, and cannot be edited by *TheraLink* while in progress. *SessionComplete* is a flag that marks if all activities in the session have been completed. The only way for a session to become complete is for the user to finish all games in their current session. *Locked_Unlocked_Status* is a flag that indicates to *TheraTouch* if the activity order is locked. The primary key for SessionOptions is *SessionID*.

4.2.6 SessionActivities

SessionActivities is used to store the relationships between sessions and their associated activities and activity options. *ActivityID* indicates which activity the record is for. *ActivityComplete* is a flag that marks if the activity has been completed in *TheraTouch*. *SessionID* is a foreign key from the SessionOptions table. *SessionID* and *ActivitySeqNum* are the primary keys for the SessionActivities table. *ActivitySeqNum* is a number which indicates which index an instance of an activity is within a session. The first activity in a session will have the index of 1; the second activity will have an index of 2 etc.

4.2.7 ActivityOptions

This table stores the options that have been specified for each instance of an activity within a session. Foreign keys *SessionID* and *ActivitySeqNum* are from the SessionActivities table. *ActivityID* is a foreign key from the Activity table. *ActivityOptionID* is not a foreign key, but corresponds to options from the ActivityOptionList table and indicates which option this record is for. *OptionValue* stores the selected value for the option, and comes from the ActivityOptionValuesEnumerated table. *SessionID*, *ActivitySeqNum*, and *ActivityOptionID* are the primary keys for this table.

4.2.8 ActivityResults

ActivityResults stores the data gathered by the *TheraTouch* framework during a session. This table has no defined relationship to other tables, but certain columns to correspond to other tables. *SessionID* and *ActivitySeqNum* indicate which session and instance of an activity within the session that the record is for. *ActivityResultID* is an artificial key needed to make each row unique. *ActivityID* indicates which activity this record is about. *ActivityRoundID* stores which round within an activity this record is about. If the data value applies to the entire activity or the activity does not have rounds, then the *ActivityRoundID* should be 0. *ActivityData* indicates what information this record is about, for example “Accuracy” or “Time Elapsed”. *DataValue* is the actual value for the ActivityData that has been collected by *TheraTouch*.

4.3 Adding an Activity

To add an activity to the database, several tables must be updated.

1. Insert a record into the Activity Table indicated the short name of the activity and the long name to be used for the various user interfaces. Also, the activity must be assigned a unique numerical *ActivityID*.

2. For each option that the activity has, a record must be added to the ActivityOptionList table providing information about the option and the default value.

3. For each possible value for the option, a record must be added to the ActivityOptionValuesEnumerated table.

4.4 SQL Examples

4.4.1 Adding Activity and Options

The following SQL statements add the Card Match activity to the *TheraTouch* database:

```
insert into [Activity] values ('CardMatch','Card Matching',2);

insert into [ActivityOptionList] values (2,4,'Difficulty','Enumerated',
'Easy(2 x 2)');

insert into [ActivityOptionValuesEnumerated] values (2,4,1,'Easy(2 x 2)');

insert into [ActivityOptionValuesEnumerated] values (2,4,2,'Medium(4 x 4)');

insert into [ActivityOptionValuesEnumerated] values (2,4,3,'Hard(6 x 6)');

insert into [ActivityOptionList] values (2,5,'Shape Types','Enumerated'
,'Fruits' );

insert into [ActivityOptionValuesEnumerated] values (2,5,1,'Fruits');

insert into [ActivityOptionValuesEnumerated] values (2,5,2,'Shapes');

insert into [ActivityOptionValuesEnumerated] values (2,5,3,'Random');

insert into [ActivityOptionList] values (2,6,'Timer Enabled','Enumerated',
'False');

insert into [ActivityOptionValuesEnumerated] values (2,6,1,'True');

insert into [ActivityOptionValuesEnumerated] values (2,6,2,'False');
```

Three tables are affected: Activity, ActivityOptionList, and ActivityOptionValuesEnumerated. The Activity table stores the names and *ActivityID* of an activity. The ActivityOptionList table stores all of the options and the *ActivityOptionID* for that activity. The ActivityOptionValuesEnumerated stores all of the option values for each option of an activity. The option values for Card Match's Difficulty option would be Easy(2 x 2), Medium(4 x 4), and

Hard(6 x 6). The option values for Card Match's Shape Types option are Fruits, Shapes, and Random. The option values for Card Match's Timer Enabled option are True and False.

These are the only statements needed for Card Match to be successfully added to the TheraDB database.

5. TheraLink Development

5.1 Introduction

TheraLink is the staff web application used to control the *TheraTouch* project. It supports the creation of sessions, the addition of users, and running reports based on session results. *TheraLink* is designed in such a way that no modifications need to be made to support the addition of activities, with the exception of creating reports specific to each activity.

5.2 Functional Area Descriptions

5.2.1 User Management

User Management contains the functionality to create users and print user tags for logging into *TheraTouch*. When a staff user presses the submit button to add a user, validation is performed to make sure the data is correct (such as having a unique Medical Record Number). After the data is verified, the user is assigned a TheraID and added to the database. The user is then shown a panel containing a representation of the TheraTag. If they choose to print the 'TheraTag', a JavaScript function is run to print the contents of the div tag that contain the 'TheraTag'.

5.2.2 Session Management

Session Management contains the functionality for creating and editing sessions. The primary view for Session Management shows each instance of an activity within a GridView. Each option for the activity is displayed in a "Sub GridView".

5.2.3 Report Management

Three types of reports were created for *TheraLink*. The View Activity Data report is a method of viewing the raw results returned by the TheraTouch framework. The table that displays this information can be filtered using the dropdown lists at the top of the table, or sorted using the sort buttons. Sorting by numerical fields does not work properly. The GridView will perform an ASCII sort instead of alphanumeric.

5.3 Common Functions

Several functions common throughout the *TheraLink* web application were moved to classes within the App_Code folder.

5.3.1 Database Class

The Database class contains several functions for retrieving information from the *TheraDB* database. When information in the database needs to be updated or removed, the web page should obtain an instance of a *SqlDataSource* object by calling Database.GetDataSource(). When information needs to be read from the database, the web page should call Database.GetDataView(). To verify that the database contains a user specified *TheraID* or *MedicalRecordNumber*, use the Database.VerifyTheraID() or Database.VerifyMRN() functions. To make sure the user associated with a *TheraID* is marked as active, use the Database.CheckIfTheraIDIsActive() function. To make sure the user associated with a *MedicalRecordNumber* is active, obtain the user's *TheraID* and then check to make sure it is active.

6. TheraTouch Development

6.1 Introduction

TheraTouch was designed to allow for developers to easily incorporate new applications into the framework without any modifications to the framework code. It also provides developers a suite of utilities that are used in the development of activities.

6.2 Framework Classes

6.2.1 Main

This class is used to manage what state the framework is in and to transition between states. The framework can be in three one of these three states: the *MainMenu* state, the *SessionController* state, or the *Activity* state. It uses the method, ChangeHealingState(), to transition between states.

6.2.1.1 ChangeHealingState()

This is the method which is called to handle transition to and from states. It will transition to whichever *HealingState* is passed to it. It is called when: a user logs into their session or when a user presses freeplay, a user logs out of their session or exits freeplay, when a user launches an activity, or when an activity exits.

6.2.2 MainMenu

MainMenu is the home screen of the framework which waits for either a user to log into their session or for a user to press freeplay. This class' major method is contactTarget_ContactAdded().

6.2.2.1 ContactTarget_ContactAdded()

This method is called whenever a contact is placed on the surface. If the contact is an identity tag, it will call the Database.GetCurrentSessionID(), to try and retrieve that user's session. It will display an error message if the *TheraID* does not exist or the user does not have an open session in the database.

6.2.3 SessionController

SessionController is the screen that displays all of the activities available in the current configuration, freeplay or session mode, and allows the user to scroll through them, select one, and then launch it. In session mode, it will determine which activities are completed and disable them. The user also has the option to launch each session mode activity as practice. This enables them to try the activity without collecting data or counting towards session completion. Upon completion of all the activities, the *SessionController* will notify the user and force them to log out. The code for scrolling is in the Update() method.

6.3 TheraTouchUtilities

To gain access to the utilities in the TheraTouchUtilities.dll, you need to add it to the reference folder of your activity. Then, you need to include this following line at the top of your code

```
using TheraTouchUtilities;
```

6.3.1 Activity

This class contains methods that need to be used in an activity and should be extended in the main class of the activity, replacing the `Microsoft.Xna.Framework.Game` extension. It defines overridable methods for pausing when an admin tag is placed on the surface, for connecting to the database, actions when the *PopupMenu* buttons are pressed, and for exiting the activity.

6.3.2 Database

This class is used for all database communication. It contains methods that are used in the framework and activities. On the framework side, it contains methods for getting a user's activities, marking a session as in progress, and logging out of a session. For activities, it contains methods to get the activity's options and submit the activity's results.

6.3.3 PopupMenu

This class is the menu which appears when the game is paused , either during freeplay mode or when an employee tag is placed on the surface during session mode. It is used by overriding the following Activity class methods: `Resume()`, `Restart()`, `Options()`, and `Quit()`. It requires that the 'resume.png', 'restart.png', 'options.png', and 'quit.png' graphics be located in a folder called 'Buttons' which should be placed within the Content folder. It also needs the graphics 'box.png' and 'cover.png'. All graphics files are available in the 'Graphics' folder on the project disk.

6.3.4 Button

This class defines a generic button that is used in the framework, the PopupMenu class, and all of the activities. To use the button, you need to add a `Button.EventHandler` to `Button.Released`, `Button.Pressed`, or both. The Released event will trigger when the user removes contact after pressing the button. The Pressed event will trigger as soon as the user presses the button.

6.3.5 RadioButton

This class defines a generic radio button which is used in the SessionController and some of the activities. To use the RadioButton, it must first be added to a RadioButtonGroup using `RadioButtonGroup.AddRadioButton()`.

6.3.6 RadioButtonGroup

This class controls all of the RadioButtons which are added to the group. There are standard methods for updating the selected RadioButton by specifying either an integer or the RadioButton itself. You can also add a `RadioButtonGroup.EventHandler` to the `SelectionUpdated` event, which is triggered whenever a RadioButton is selected.

6.4 Activities

6.4.1 Creating a new Activity

1. To create a new activity, open up Microsoft Visual Studio 2008 and create a new project using the Surface Application (XNA Game Studio 3.0) template. Be sure to give your activity the name which it will be called in *TheraDB* and *TheraLink*. The names need to match in order for the framework to launch the activity.
2. Right Click on the Reference folder and then click Add Reference. Locate and add the 'TheraTouchUtilities.dll'. Once added, include the following using directives:

```
using Microsoft.Surface;
using Microsoft.Surface.Core;
using TheraTouchUtilities;
```

3. To give your activity an icon, create a 200x200 .png called icon.png and add it to the project folder. Do not put it in the Content folder, and make sure it gets built when the activity compiles by changing the copy property to read 'Copy If Newer'.
4. In the main class (by default named `App1`), replace the `Microsoft.Xna.Framework.Game` extension with `Activity`. This will give you the access to the necessary methods in the Activity class.
5. You will need two constructors. One should take no parameters and it will be the Freeplay Mode constructor. The other will be the Session Mode constructor and it should take the following parameters: int sessionID, int activitySeqNum, and bool practice. SessionID and activitySeqNum are used in the Database method GetActivityOptions and practice is used to determine whether the results will be submitted at the end of the activity.
6. To specify which of the constructors gets called, open the Program class and edit the main method to mirror the following code. The following code is assuming the default class names.

```
Application.SetUnhandledExceptionMode(UnhandledExceptionMode.ThrowException);

GlobalizationSettings.ApplyToCurrentThread();

if (args.Length == 3)
   {
      using(App1 app = new App1(int.Parse(args[0]), int.Parse(args[1]),
      bool.Parse(args[2])))
      {
          app.Run();
      }
   }
```

```
else
    {
        using (App1 app = new App1())
        {
            app.Run();
        }
    }
```

7. When executing an activity in Session Mode, the application needs to connect to the database before attempting to retrieve activity options. Insert the following code to retrieve options:

```
try
{
    ConnectDatabase();
    options = Database.GetActivityOptions(sID, sNum);

}
catch (Exception)
{
    ConnectionLost = true;
    ExitActivity();
}
```

`GetActivityOptions()` will return a List of string arrays which contains the option name and value. You will also do this for the SubmitQuery method (see #9 below).

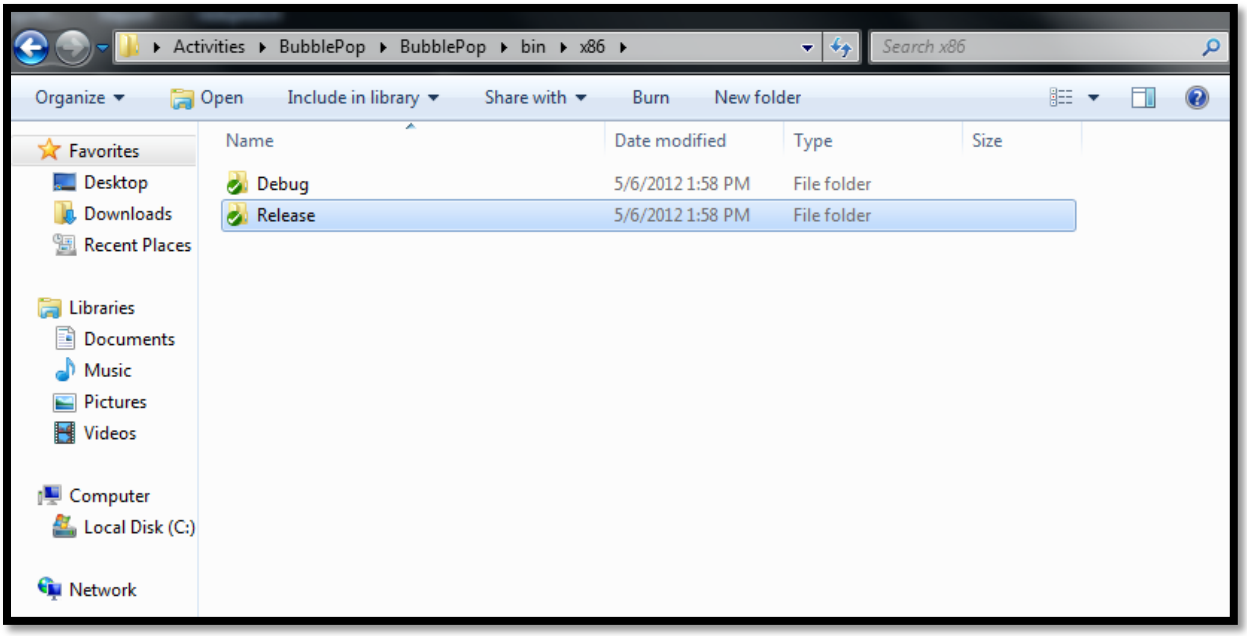
8. Use CurrentActivityState to determine what ActivityState the activity is in and ChangeActivityState method to transition between states. The ActivityStates are: DEMO, END, MAIN_MENU, and PLAY. To use ChangeActivityState, simply override the method, and specify what occurs with each state change. The CurrentActivityState will determine if the employee tag can pause the game.
9. In Session Mode, once the game is completed and the results are submitted to the database, set Complete = true, and call ExitActivity().

```
try
{
    this.Database.SubmitQuery(sID, sNum, results);
    Complete = true;
}
catch (Exception)
{
   ConnectionLost = true;
   ExitActivity();
}
```

6.4.2 Deploying the Activity

Once the activity is complete and ready to be incorporated into the framework, take the following steps to deploy the activity.

1. Navigate to the 'C:\Program Files\TheraTouch\Activities' folder and create a new folder with the same name as your activity. It must be named exactly how it is named in *TheraDB* and the activity's '.exe'.
2. Copy the output of your activity's build, the .exe, Content, 'icon.png', etc, into the newly created folder.



3. The framework will now recognize your activity in Freeplay Mode and Session Mode. It will not be playable in Session Mode until you have added the activity to *TheraDB* and created a session.

7. Appendix

1. Deploying the framework application to the Microsoft Surface, http://msdn.microsoft.com/en-us/library/ee804847(v=surface.10).aspx

2. Setting the Microsoft Surface to Single Application Mode, http://msdn.microsoft.com/en-us/library/ee804972(v=surface.10).aspx