Project #4 - Remote Code Publisher

Version 1.4, Revised: 04/12/2017 17:49:53
Due Date: Tuesday May 2nd

Purpose:

A Code Repository is a Program responsible for managing source code resources, e.g., files and documents. A fully developed Repository will support file persistance, managment of versions, and the acquisition and publication of source and document files. A Remote Repository adds the capability to access the Repository's functionality over a communication channel, e.g., interprocess communication, inter-network communication, and communication across the internet.

In this project we will focus on the publication functionality of a Remote Repository. We will develop a remote code publisher, local client, and communication channel that supports client access to the publisher from any internet enabled processor.

The communication channel will use sockets and support an HTTP like message structure. The channel will support:

- HTTP style request/response transactions
- One-way communication, allowing asynchronous messaging between any two endpoints that are capable of listening for connection requests and connecting to a remote listener.
- Transmission of byte streams that are set up with one or more negotiation messages followed by transmission of a stream of bytes of specified stream size[2].

The Remote Code Publisher will:

- Support publishing web pages that are small wrappers around C++ source code files, just as we did in Project #3.
- Accept source code text files, sent from a local client.
- Support building dependency relationships between code files saved in specific repository folders, based on the functionality you provided in Project #2 and used in Project #3.
- Support HTML file creation for all the files in a specified repository folder[1], including linking information that displays dependency relationships, and supports and navigation based on dependency relationships.
- Delete stored files, as requested by a local client.

Clients of the Remote Code Publisher will provide a Graphical User Interface (GUI) with means to:

- Upload one or more source code text files to the Remote Publisher, specifying a category with which those files are associated[1].
- Display file categories, based on the directory structure supported by the Repository.
- Display all the files in any category.
- Display all of the files in any category that have no parents.
- Display the web page for any file in that file list by clicking within a GUI control. This implies that the client will download the appropriate webpages, scripts, and style sheets and display, by starting a browser with a file cited on the command line[2].
- On starting, will download style sheet and JavaScript files from the Repository.

Note that your client does not need to supply the functionality to display web pages. It simply starts a browser to do that. Browsers will accept a file name, which probably includes a relative path to display a web page from the local directory.

You could also start IIS web server and provide an appropriate URL to the browser on startup. Either approach is acceptable. If you use IIS, you won't have to download files, but you are obligated to show that you can do that.

Requirements:

Your Remote Repository:

1. (**2**) **Shall** use Visual Studio 2015 and its C++ Windows console projects, as provided in the ECS computer labs. You must also use Windows Presentation Foundation (WPF) to provide a required client Graphical User Interface (GUI).

2. (**1**) **Shall** use the C++ standard library's streams for all console I/O and new and delete for all heap-based memory management.

3. (**3**) **Shall** provide a Repository program that provides functionality to publish, as linked web pages, the contents of a set of C++ source code files.

4. (**4**) **Shall**, for the publishing process, satisfy the requirements of CodePublisher developed in Project #3.

5. (**4**) **Shall** provide a Client program that can upload files[3], and view Repository contents, as described in the Purpose section, above.

6. (**3**) **Shall** provide a message-passing communication system, based on Sockets, used to access the Repository's functionality from another process or machine.

T N P B

7. (**2**) The communication system **shall** provide support for passing HTTP style messages using either synchronous request/response or asynchronous one-way messaging.

8. (**1**) The communication system **shall** also support sending and receiving streams of bytes[6]. Streams will be established with an initial exchange of messages.

9. (**5**) **Shall** include an automated unit test suite that demonstrates you meet all the requirements of this project[4] including the transmission of files.

10. (**5 point bonus**) **Shall optionally** use a lazy download strategy, that, when presented with a name of a source code web page, will download that file and all the files it links to. This allows you to demonstrate your project using local webpages instead of downloading the entire contents of the Code Publisher for demonstration.

11. (**5 point bonus**) **Shall optionally** have the publisher accept a path, on the commandline, to a virtual directory on the server. Then support browsing directly from the server by supplying a url to that path when you start a browser. This works only if you setup IIS on your machine and make the path a virtual directory. The TAs will do that on the grading machines.

---

1. Categories are the names of folders in which the Repository stores its source code and web files. You may define Categories in any way that seems sensible. For example, they could simply be the namespace(s) for the uploaded files, or a Client supplied name.

2. You will find a demonstration of how to programmatically start an application here.

3. The stream capablity is intended to send files, which could be either text or binary format. Stream size will be the file size.

4. Transmitting and receiving byte streams will be used to send and receive files in either text or binary format.

5. This is in addition to the construction tests you include as part of every package you submit.

What you need to know:

In order to successfully meet these requirements you will need to know:

1. Details of the C++ language: http://CppReference.com including C++11 threading and concurrency models.

2. C++\CLI or C# syntax in order to build the Client GUI.

3. How sockets and the provided socket library work.

4. All those things you learned while developing code for Projects #1, #2, and #3.

Back Back