|  |
| --- |
| SAN FRANCISCO COORDINATING CENTER |
| Code Sample Descriptions |
|  |
| *Revision 1* |
| **Produced by Remi Frazier** |
| **11/8/2013** |

|  |
| --- |
|  |

# Intuit/Quickbooks Paypage Integration

## Description

At the end of 2011, a web services company contacted me seeking assistance with a project they had been struggling with in-house for over a month. One of their products relied on accepting payments through the online Quickbooks PayPage system, an unsupported legacy service offered by Intuit. My client had been unable to successfully integrate their existing ASP Classic site with the PayPage product, and as a result were unable to collect payments from users. The situation was desperate – my client had a contractual obligation to take the system live less than a week after they turned to me.

## Why it’s cool

My job was complicated by the client’s security posture, which prevented me from seeing any part of their existing codebase. Despite this, I was able to troubleshoot and resolve the issue with a very fast turnaround (over the course of a weekend), employing undocumented features of the Intuit system, and deliver clean code to the client which was ready to drop into production and deployed immediately. I have since worked with this client to help update their codebase for this site to ASP.NET, but I’m still the most proud of this original code block.

# SQL Server Regular Expression Processing

## Description

In 2010, one of my long-term clients requested an extra layer of protection against possible injection attacks against their MSSQL servers. They had suffered several security breaches over a five year period, and wanted to explore methods for mitigating future attacks. In particular, they were seeking a method to positively validate inserted/updated strings on their database servers to ensure that even if data validation on the web server was compromised, injection attacks would not succeed.

The solution I produced involved the use of stored procedures for all inserts/updates, employing regular expression pattern matching to enforce string validation. Because SQL Server does not natively support regular expression processing, this required the deployment of a C# assembly to the database engine.

## Why it’s cool

This was a case where a seemingly minor (and non-novel) solution to a problem yielded surprising results. Once pattern matching was available on their database servers, DBAs in the client’s organization started employing regular expressions to refine their daily queries. This led to established in-house developers requesting training from me and then deploying regexps in their daily work (which they had not been previously), which produced significantly cleaner and faster software. I’m still proud of this simple piece of code, because it showed me how small changes or features can produce disproportionately large results in an entire business process.