Performance Benchmark Test

This document provides simple benchmark tests you can run to compare the Original vs Optimized versions.

🧪 Quick Benchmark Tests

Test 1: Logging Performance

Add this to your RDLC Custom Code to test logging performance:

```
' Benchmark: Log 100 messages
Public Function BenchmarkLogging() As String
    Dim startTime As DateTime = DateTime.Now
    Dim i As Integer

    For i = 1 To 100
        WriteLog("Test message number " & i.ToString())
    Next

    Dim endTime As DateTime = DateTime.Now
    Dim elapsed As TimeSpan = endTime.Subtract(startTime)

    Return "Logged 100 messages in: " & elapsed.TotalMilliseconds.ToString() &
" ms"
End Function
```

How to use:

1. Add function to Custom Code
2. Add textbox with expression: `=Code.BenchmarkLogging()'`
3. Run report and check result

Expected Results:

- Original: ~500ms
- Optimized: ~100ms

Test 2: String Concatenation Performance

```
' Benchmark: Concatenate 50 strings
Public Function BenchmarkStringConcat() As String
    Dim startTime As DateTime = DateTime.Now
    Dim testStrings(49) As String
    Dim i As Integer
    ' Fill array
    For i = 0 To 49
        testStrings(i) = "Line number " & i.ToString()
    Next

    ' Concatenate 10 times
    For i = 1 To 10
        Dim result As String = ConcatenateNonEmptyWithCrLf(testStrings)
    Next

    Dim endTime As DateTime = DateTime.Now
    Dim elapsed As TimeSpan = endTime.Subtract(startTime)

    Return "Concatenated 50 strings x 10 in: " &
elapsed.TotalMilliseconds.ToString() & " ms"
End Function
```

Expected Results:

- Original: ~120ms
- Optimized: ~30ms

Test 3: Number to Words Performance

```
' Benchmark: Convert numbers to words with repetition
Public Function BenchmarkNumberToWords() As String
    Dim startTime As DateTime = DateTime.Now
    Dim result As String = ""
    Dim i As Integer

    ' Convert same numbers multiple times (tests cache)
    For i = 1 To 20
        result = ToWordsIn(1000)  ' Same number
        result = ToWordsIn(5000)  ' Same number
        result = ToWordsIn(10000) ' Same number
        result = ToWordsIn(25000) ' Different numbers
        result = ToWordsIn(50000)
    Next

    Dim endTime As DateTime = DateTime.Now
    Dim elapsed As TimeSpan = endTime.Subtract(startTime)

    Return "Converted 100 numbers in: " & elapsed.TotalMilliseconds.ToString()
& " ms"
End Function
```

Expected Results:

- Original: ~250ms (no caching)
- Optimized: ~50ms (with caching)

Test 4: Key-Value Parsing Performance

```vb
' Benchmark: Parse key-value lists
Public Function BenchmarkKeyValueParsing() As String
    Dim startTime As DateTime = DateTime.Now
    Dim testData As String = ""
    Dim i As Integer

    ' Create test data: 50 key-value pairs
    For i = 1 To 50
        If testData <> "" Then testData &= Chr(177)
        testData &= "Key" & i.ToString() & Chr(177) & "Value" & i.ToString()
    Next

    ' Parse 20 times
    For i = 1 To 20
        Dim tempDict As Object = Nothing
        SetDataAsKeyValueList(tempDict, testData)
    Next

    Dim endTime As DateTime = DateTime.Now
    Dim elapsed As TimeSpan = endTime.Subtract(startTime)

    Return "Parsed 50 pairs x 20 in: " & elapsed.TotalMilliseconds.ToString() &
" ms"
End Function
```

Expected Results:

- Original: ~800ms (O(n^2) algorithm)
- Optimized: ~400ms (O(n) algorithm)

📊 Complete Benchmark Suite

Add all four functions and create a simple report:

```vb
' Complete benchmark suite
Public Function RunAllBenchmarks() As String
    Dim sb As New System.Text.StringBuilder()

    sb.AppendLine("=== PERFORMANCE BENCHMARK RESULTS ===")
    sb.AppendLine("")
    sb.AppendLine("Test 1 - Logging:")
    sb.AppendLine(BenchmarkLogging())
    sb.AppendLine("")
    sb.AppendLine("Test 2 - String Concatenation:")
    sb.AppendLine(BenchmarkStringConcat())
    sb.AppendLine("")
    sb.AppendLine("Test 3 - Number to Words:")
    sb.AppendLine(BenchmarkNumberToWords())
    sb.AppendLine("")
    sb.AppendLine("Test 4 - Key-Value Parsing:")
    sb.AppendLine(BenchmarkKeyValueParsing())
    sb.AppendLine("")
    sb.AppendLine("=== END BENCHMARK ===")

    Return sb.ToString()
End Function
```

🎯 Real-World Benchmark

Setup Instructions

1. Create a simple RDLC report with a dataset of 1000 rows
2. Add these expressions in detail section:

```
' Company name retrieval
=Code.GetVal("CompanyName")'

' Number to words
=Code.ToWordsIn(Fields!Amount.Value, True, True)'

' String concatenation
=Code.ConcatenateNonEmptyWithCrLf(New String() {
    Fields!Line1.Value,
    Fields!Line2.Value,
    Fields!Line3.Value
})'

' Logging
=Code.WriteLog("Processing row: " & Fields!RowID.Value)'
```

3. Run report and measure execution time

📈 Benchmark Results Template

Copy this template to record your results:

```
BENCHMARK RESULTS
Date: _______________
```

```
Environment: _______________
Dataset Size: _______________

                          | Original | Optimized | Improvement
--------------------------|----------|-----------|-------------
Logging (100 calls)       |    ms    |     ms    |         %
String Concat (50x10)     |    ms    |     ms    |         %
Number to Words (100)     |    ms    |     ms    |         %
Key-Value Parse (50x20)   |    ms    |     ms    |         %
--------------------------|----------|-----------|-------------
Real Report (1000 rows)   |    sec   |     sec   |         %
Memory Usage              |    MB    |     MB    |         %

Notes:
_________________________________________________________________
_________________________________________________________________
_________________________________________________________________
```

💡 Interpreting Results

Good Performance Gains

If you see improvements like these, the optimization is working well:

```
Logging:           80%+ faster
String Concat:     70%+ faster
Number to Words:   60%+ faster (with cache hits)
Key-Value Parse:   40%+ faster
Overall Report:    40-60% faster
```

Marginal Gains

If improvements are < 20%, consider:

- Dataset may be too small to show benefits
- Report may be I/O bound (database query slow)
- Most time spent in rendering, not custom code
- Custom code not being called frequently

No Improvement

If you see no improvement:

1. **Check you're using optimized version**
    - Verify `RdlcReportCode_Optimized.vb` is in Custom Code
2. **Verify custom code is being called**

    - Add logging to confirm functions execute

3. **Check for external bottlenecks**

    - Database query performance
    - Network latency
    - Report rendering engine

🔍 Detailed Profiling

Method 1: Log Timing

Add timestamp logging:

```
Public Function TimedGetVal(key As Object) As Object
    Dim startTime As DateTime = DateTime.Now
    Dim result = GetVal(key)
    Dim elapsed As TimeSpan = DateTime.Now.Subtract(startTime)

    WriteLog("GetVal(" & key.ToString() & ") took: " &
elapsed.TotalMilliseconds.ToString() & "ms")
    Return result
End Function
```

Method 2: Use Stopwatch

For more accurate timing:

```
Public Function BenchmarkWithStopwatch() As String
    Dim sw As New System.Diagnostics.Stopwatch()

    sw.Start()
    ' Your code to benchmark
    For i = 1 To 100
        WriteLog("Test")
    Next
    sw.Stop()

    Return "Elapsed: " & sw.ElapsedMilliseconds.ToString() & " ms"
End Function
```

Method 3: Memory Profiling

Check memory usage:

```
Public Function CheckMemoryUsage() As String
    Dim beforeMem As Long = GC.GetTotalMemory(False)

    ' Your memory-intensive operation
    Dim testArray(999) As String
    For i = 0 To 999
        testArray(i) = "Test string " & i.ToString()
    Next
    Dim result = ConcatenateNonEmptyWithCrLf(testArray)

    Dim afterMem As Long = GC.GetTotalMemory(False)
    Dim usedMem As Long = afterMem - beforeMem

    Return "Memory used: " & (usedMem / 1024).ToString() & " KB"
End Function
```

📝 Benchmark Best Practices

DO:

☑ Run benchmarks multiple times and average results
☑ Use realistic data volumes
☑ Clear caches between tests when comparing
☑ Test in similar environment to production
☑ Document test conditions

DON'T:

❌ Benchmark with < 10 iterations
❌ Compare different environments
❌ Test with trivial data (< 10 rows)
❌ Ignore warm-up runs
❌ Forget to account for caching

🎓 Advanced Benchmarking

Create Comparative Report

RDLC Report Structure:

```
[Header]
- Report title
- Benchmark date/time

[Body - Test Results Table]
Columns:
```

```
- Test Name
- Original Time
- Optimized Time
- Improvement %

[Footer]
- Total execution time
- Memory usage
- Conclusion
```

Automated Benchmark

```
Public Function AutomatedBenchmark() As String
    Dim results As New System.Text.StringBuilder()

    results.AppendLine("AUTOMATED BENCHMARK REPORT")
    results.AppendLine("Date: " & DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"))
    results.AppendLine("")

    ' Run each test multiple times
    Dim iterations As Integer = 5
    Dim i As Integer

    ' Test 1: Logging
    Dim logTotal As Long = 0
    For i = 1 To iterations
        Dim sw As New System.Diagnostics.Stopwatch()
        sw.Start()
        BenchmarkLogging()
        sw.Stop()
        logTotal += sw.ElapsedMilliseconds
    Next
    results.AppendLine("Logging (avg of " & iterations & "): " & (logTotal /
iterations).ToString() & " ms")

    ' Add more tests...

    Return results.ToString()
End Function
```

📞 Support

If your benchmark results differ significantly from expected results:

1. Verify you're using the correct file version
2. Check that test data is appropriate size
3. Ensure environment is consistent
4. Review log files for errors

5. Compare with baseline results in `OPTIMIZATION_GUIDE.md`

🎯 Quick Checklist

Before running benchmarks:

- [ ] Backup current RDLC file
- [ ] Close other applications (for consistent results)
- [ ] Use realistic test data size
- [ ] Clear cache before each test run
- [ ] Document test environment
- [ ] Run multiple iterations
- [ ] Calculate average results
- [ ] Compare with expected benchmarks
- [ ] Document any anomalies
- [ ] Save results for future comparison