APPENDIX F

DEVELOPER DOCUMENTATION

INTRODUCTION

This appendix outlines procedures that software developers may use in implementing Process 12-50, and provides a brief background. Because each program or tool is designed and written differently, no single "correct" procedure for implementation exists. The goals of this document are to present the basic structure behind Results comparisons and the primary steps involved so that the software developer can easily make decisions on how Process 12-50 can best be implemented in their software. Examples using different development languages are also provided. The implementation details are left to the developers.

PROCESS 12-50 OVERVIEW

Process 12-50 is conceptually simple:

1. Understand the problem domain associated with an engineering computational process (CP).
2. Divide the large problem domain into smaller subdomains as necessary. For relatively simple computations, this division may not be required. The "bridge engineering computational domain" is huge and will be divided into smaller subdomains as illustrated herein.
3. Parameterize the subdomain. Each subdomain will be described in a general sense, outlining the purpose of the computation, the limitations and scope of the computation, and finally, the parametric definition of the inputs and outputs. The assumptions associated with the computational process must be clearly outlined as well. In short, inputs are mapped to outputs in a systematic manner.
4. Create a document that explains the test suite. This document will provide a description of the test suite, assumptions for the development of the test suite, the processes that are being compared, and a discussion of the results.
5. Generate problems using the parametric form. Each computational subdomain has different functions and characteristics. For example, in the bridge engineering domain, many computations associated with structural analysis are founded on structural mechanics. Other computations are associated with AASHTO specification articles, which may be based on a combination of empirical results, mechanics, experience, construction considerations, etc. The characteristics of the subdomain must be understood and the subdomain creator (subdomain designer) must be able to design input data in the parametric form that will address the majority of issues (checks) associated with a specific subdomain. This activity can be the most challenging subdomain design task. For example, if a subdomain is designed to check prestressed concrete design/review for the *LRFD Specifications*, then all of the branches of the specifications and all of the practical parametric geometry and material properties need to be addressed. Moreover, the process should provide proof that various articles are checked/validated.
6. Perform the computation for problems within the subdomain and produce results in a specified format. The execution of this task depends on the implementation of the computation within software or hand processes. For example, CPs may perform computations by hand, spreadsheet, macros, and traditional programs. The implementation may be batch oriented or highly interactive. Hence, the approach will vary. For all implementations, the parametric data outlined in Step 4 above must be entered into the program/process and executed.
7. Examine the results from the computations. The examination of results will depend on the developer's intent and can vary widely. The examination might include, but is not limited to:
   - Creation of a test suite for program validation,
   - Comparison with other CPs, including hand computations,
   - Development of plots illustrating trends in input vs. output parameters,
   - Development of plots illustrating trends in output vs. location, and
   - Development of plots illustrating trends in output vs. various methods, assumptions, specification versions, state specifications, etc.
8. Create a process for examination. The examination of results can be a laborious task and should be automated whenever appropriate to do so. For example, in this research, the focus was to develop a process (i.e., Process 12-50) for checking software computations directed at the *LRFD Specifications*. In this development, hundreds of problems were parameterized for numerous subdomains. These data could be systematically processed with numerous approaches, several of which are outlined in this report. Note that Process 12-50 does not depend on the tools used for input generation nor the examination of results. This is critical, because tools will change, the results will change, but the

underlying process for parameterization should remain relatively constant.

This appendix will focus on Step 5 from the list above, performing the computations for the problems within a subdomain and producing the results in the Process 12-50 results format. This is the part of Process 12-50 that software developers would implement within their native programs for a subdomain that has been previously defined. In order to better understand the goals of Process 12-50, Figure F-1 outlines the database structure currently used to review results. All of the tables in the database structure are broken out into three categories:

1. Required tables that must be produced dynamically by each program, computational engine, or other CPs,
2. Required tables that are previously filled with static data, or
3. Optional tables.

Note that a comma-delimited text file can be imported into a database as a table. Therefore, in this document, the terms file and table can be used almost interchangeably. *File* typically refers to the comma-delimited text output generated by a computational engine, while *table* refers to the same data after it has been imported into a database.

Each computation engine must produce only one table dynamically when it is executed—the Results table. This table contains the results of the computations. All other tables contain information about the results. The Results table contains the fields outlined in Table F-1. Note that for a comma-delimited text file, one field represents everything between two commas.

A short sample of the Results text file that is imported to create the Results table is shown in Figure F-2.

If the Results table is added to a database in addition to the fixed tables, results can be reviewed, along with important information about the results, such as which specification articles involve a certain result.

For additional functionality in comparing results, some or all of the optional tables from Figure F-1 can be generated. The only one of these tables with a fixed structure is the Point-of-Interest (POI) table. The POI table contains the fields outlined below in Table F-2.

The purpose of the POI table is to provide information about why a certain point is of interest. It contains a series of boolean fields that indicate whether a certain condition exists at the given point. For example, this table will indicate that a point is the location where the flange of a steel girder changes properties.

The other optional tables shown in Figure F-1 represent one possible use of the Auxiliary ID in the Results table. The intent of the Auxiliary ID is to provide flexibility for individual testers to implement additional attributes. In the example in Figure F-1, the Auxiliary ID is used to point to a wealth of information about a result, including the limit state used to produce it, the sense of the result, the type of action it represents, whether it is a maximum value, minimum value, or neither, etc. This is just one example of how the Auxiliary ID has been implemented to provide more information for pier results.

A short sample of the POI text file that is imported to create the POI table is shown in Figure F-3.

IMPLEMENTATION OF THE PROCESS

This section will examine how to implement Process 12-50 in a CP. Some of the main steps needed to accomplish this are as follows:

1. Read the NCHRP Process 12-50 final report,
2. Review the sample code for process implementation (see the subsections Visual Basic Example, FORTRAN Example, and so forth),
3. Review the Report IDs for the specific subdomain being implemented,
4. Determine which Report IDs are most important,
5. Determine which Report IDs can be written easily,
6. Determine the effort needed to write additional Report IDs that cannot be obtained easily,
7. Decide which Report IDs will be written,
8. Determine where the program will output the results, i.e.:
   a. Write from multiple locations as results are computed,
   b. Store results as they are computed and then write them all at once later, or
   c. A combination of a and b above.
9. Determine which POI data can be written,
10. Implement code to write the POI data (see Figure F-4),
11. Implement code to write Results data (see Figure F-4),
12. Test this Process 12-50 implementation, and
13. Test this newly developed software on the data on the accompanying CD-ROMs.

Each of these steps is described below.

1. Read the NCHRP Process 12-50 Final Report

This report addresses all of the issues discussed above in depth. A deeper understanding of the project will be gained by reading this report. The report also contains some sample tools for viewing/comparing results and generating test data sets. These tools lend ideas as to how one might implement testing outside of the main Process 12-50 database. The tools are based on Microsoft Office, but the concepts behind them are fundamentally simple, and they can be modified to suit specific purposes or migrated to a different platform.

2. Review the Sample Code

Sample code for a Process 12-50 implementation is shown for several programming languages (Visual Basic, FORTRAN, and Visual C++).

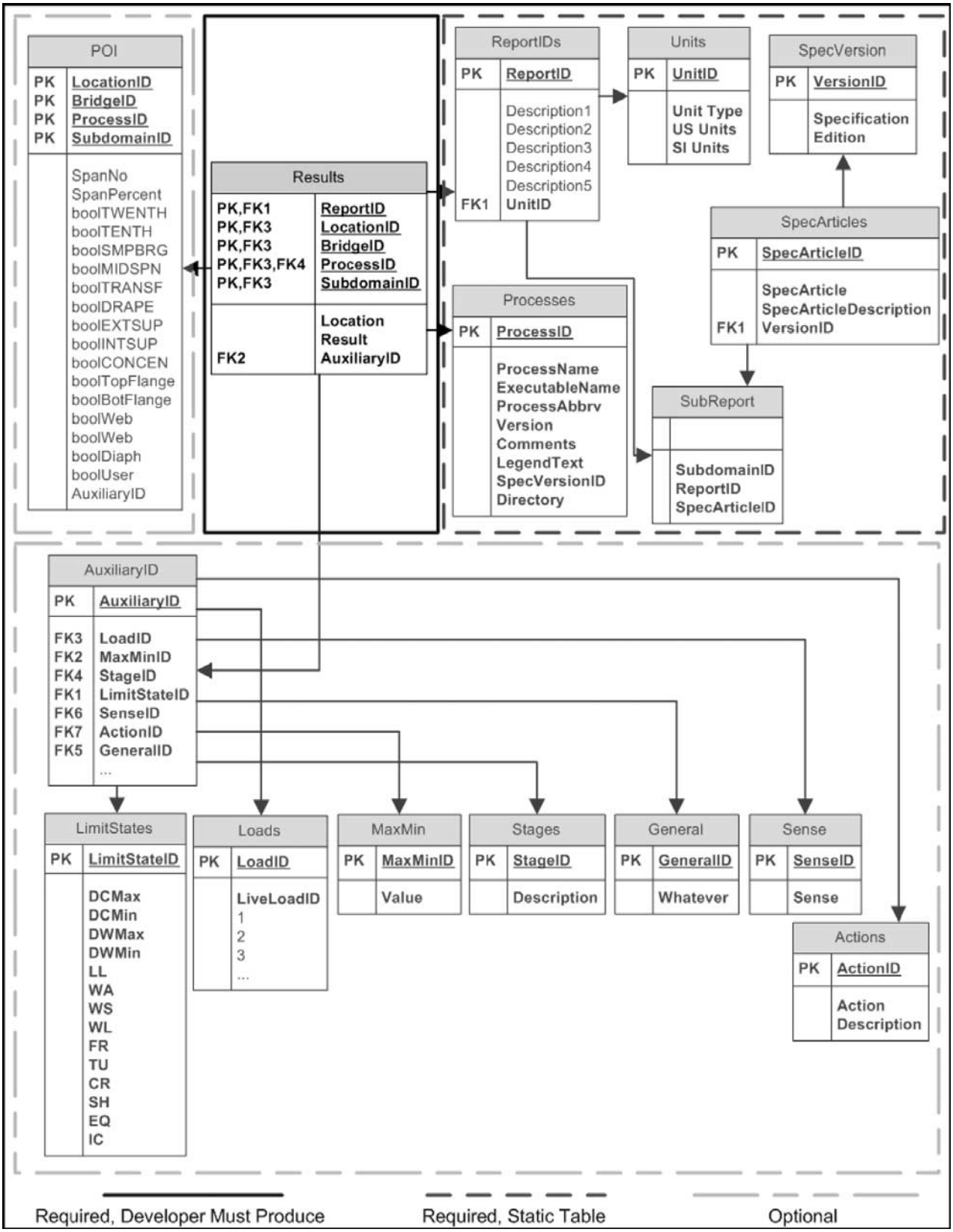

Figure F-1. Database structure.

TABLE F-1 Results table fields

Name	Description	Data Type
BridgeID	Unique identifier of an input set within a subdomain	Integer
ProcessID	Unique identifier of a computational engine	Integer
ReportID	Unique identifier for an output value or group of values	Integer
Location	Location of the result	Real
Result	Actual result	Real
SubdomainID	Unique identifier of the subdomain containing the result	Integer
LocationID	Unique identifier for a point on a bridge	Integer
AuxiliaryID	Additional identifier to add flexibility to the process	Integer

```
1              ,   9, 25028,   0.000000    ,   0.000000    ,   3,    1
1              ,   9, 25029,   0.000000    ,  0.6149535E+10,   3,    1
1              ,   9, 25028,   0.000000    ,   0.000000    ,   3,    1
1              ,   9, 25029,   0.000000    ,  0.6149535E+10,   3,    1
1              ,   9, 25028,   33528.00    ,   0.000000    ,   3,   14
1              ,   9, 25029,   33528.00    ,  0.6149535E+10,   3,   14
1              ,   9, 25028,   33528.00    ,   0.000000    ,   3,   14
1              ,   9, 25029,   33528.00    ,  0.6149535E+10,   3,   14
1              ,   9, 25028,   70104.00    ,   0.000000    ,   3,   27
1              ,   9, 25029,   70104.00    ,  0.6149535E+10,   3,   27
1              ,   9, 25028,   70104.00    ,   0.000000    ,   3,   27
1              ,   9, 25029,   70104.00    ,  0.6149535E+10,   3,   27
1              ,   9, 25028,   103632.0    ,   0.000000    ,   3,   39
1              ,   9, 25029,   103632.0    ,  0.6149535E+10,   3,   39
1              ,   9, 25028,   103632.0    ,   0.000000    ,   3,   39
1              ,   9, 25029,   103632.0    ,  0.6149535E+10,   3,   39
```

Figure F-2. Example of results file.

TABLE F-2 POI table fields

Name	Description	Data Type
LocationID	To match the location ID provided in the results record.	Integer
BridgeID	Same as results record	↓
ProcessID	“ “	↓
SpanNo	Span number for this location	↓
SpanPercent	Percentage along the span.	Real
SubDomain	Bridge subdomain	Integer
boolTWENTH	Twentieth point	Boolean
boolTENTH	Tenth point	↓
boolSMPBRG	Simple span bearing location	↓
boolMIDSPN	Midspan of the simple span	↓
boolTRANSF	Prestressing transfer length	↓
boolDRAPE	Drape point location	↓
boolEXTSUP	Exterior or end support	↓
boolINTSUP	Interior support	↓
boolCONCEN	Concentrated load location	↓
boolTopFlange	Top flange cutoff	↓
boolBotFlange	Bottom flange cutoff	↓
boolWeb	Web cutoff	↓
boolDiaph	Diaphragm location	↓
boolUser	User defined	↓

```
1,      1,      2,   1,      0.0000000E+00,        3, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
2,      1,      2,   1,      0.8947746E+00,        3, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
3,      1,      2,   1,      0.5000000E+01,        3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
4,      1,      2,   1,      0.1000000E+02,        3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
5,      1,      2,   1,      0.1500000E+02,        3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Figure F-3. Example of POI file.

3. Review the Report IDs

Currently, each subdomain represents a separate test suite that examines a certain area of bridge engineering (such as multiple-span, composite, prestressed I-sections). Each subdomain has a set of Report IDs representing most of the important results that are associated with that type of calculation. Due to the differences between software designs, most programs will not be able to output all of the Report IDs for a subdomain.

4. Determine Which Report IDs Are Most Important

After reviewing the Report IDs for a subdomain, determine which IDs are most important for testing the software. For example, the results of some intermediate calculation may not indicate a problem, but the results of another intermediate calculation may flag a potential problem.

5. Determine Which Report IDs Can Be Written Easily

At some point in the code easy access is probably available to most of the results to be written to the Process 12-50 Results file. Make a list of the Report IDs that can be written without coding any additional computations.

6. Determine the Effort Needed To Write Additional Report IDs

Perhaps easy access is not available to all of the Report IDs that seem important. Some of the additional Report IDs may require the performance of additional calculations or more intermediate steps. Some Report IDs may not be available without restructuring the code entirely. Don't panic, read on.

7. Decide Which Report IDs Will Be Written

Based on the three previous steps, decide which Report IDs will be output. This decision can be made by balancing the ease of outputting a certain Report ID with its importance to specific testing needs. Process 12-50 does not require that all Report IDs be written out, and there are no existing engines that write all of the Report IDs. This step may be iterative in that the developer may start by writing only the most important Report IDs and, after comparing results, determine that more of the intermediate results are needed to explain why differences are occurring in various results. The need to explain such differences may warrant additional effort in writing the additional data.

8. Determine Where the Program Will Output the Results

While planning which Report IDs to write, perhaps investigate another key issue: when to write the Process 12-50 output. Results can be output as results are computed, or they can be stored and then written together at the end of the program. Depending on the structure of the program, one method may have a distinct advantage over the other. The main advantage to the first method is that there is no need to store a large amount of results in memory. The main advantage to the second method is that all Process 12-50 output code is localized in one area of the program. Note that if the information is stored for output later, all of the information about a particular result must be carried along with it throughout the program. Many programs already have such a storage mechanism in place for processing and output.

In general, it seems that the first method is usually better suited for most applications, but this may not always be the case. The code examples provided in the subsections regarding Visual Basic, FORTRAN, and C++, show an implementation of each of these methods. For these examples, the code is so simple that there is really no difference between the two methods, but for larger, more complex programs, it may not always be possible to know the current Report ID, Auxiliary ID, etc., at a particular location in the program.

9. Determine Which POI Data Can Be Written

By following the same steps used to determine which Report IDs can be written, it is possible to determine which POI data can be written. Some programs may not have stored all of the desired information about all of the locations when the Process 12-50 results are written. Use the same criteria that were used with the Report IDs to determine how much effort must be expended to get POI data.

10. Implement the Code to Write the POI Data

In many cases, a program will loop/increment over a set of points to write output or perform computations. This arrangement works nicely for incrementing POI data and writing it to the POI file. It is probably advisable to update and write the POI data as soon as the point is incremented. See Figure F-4 for a general overview of the implementation.

11. Implement the Code to Write the Results Data

It is probably best to place this code inside the POI loop/increment if possible, since the Location ID can be written to both files at the same time. The attached examples use the method of looping over the desired locations to write both the POI data and the Results data. Note that this data can be written in any order that is convenient for the developer.

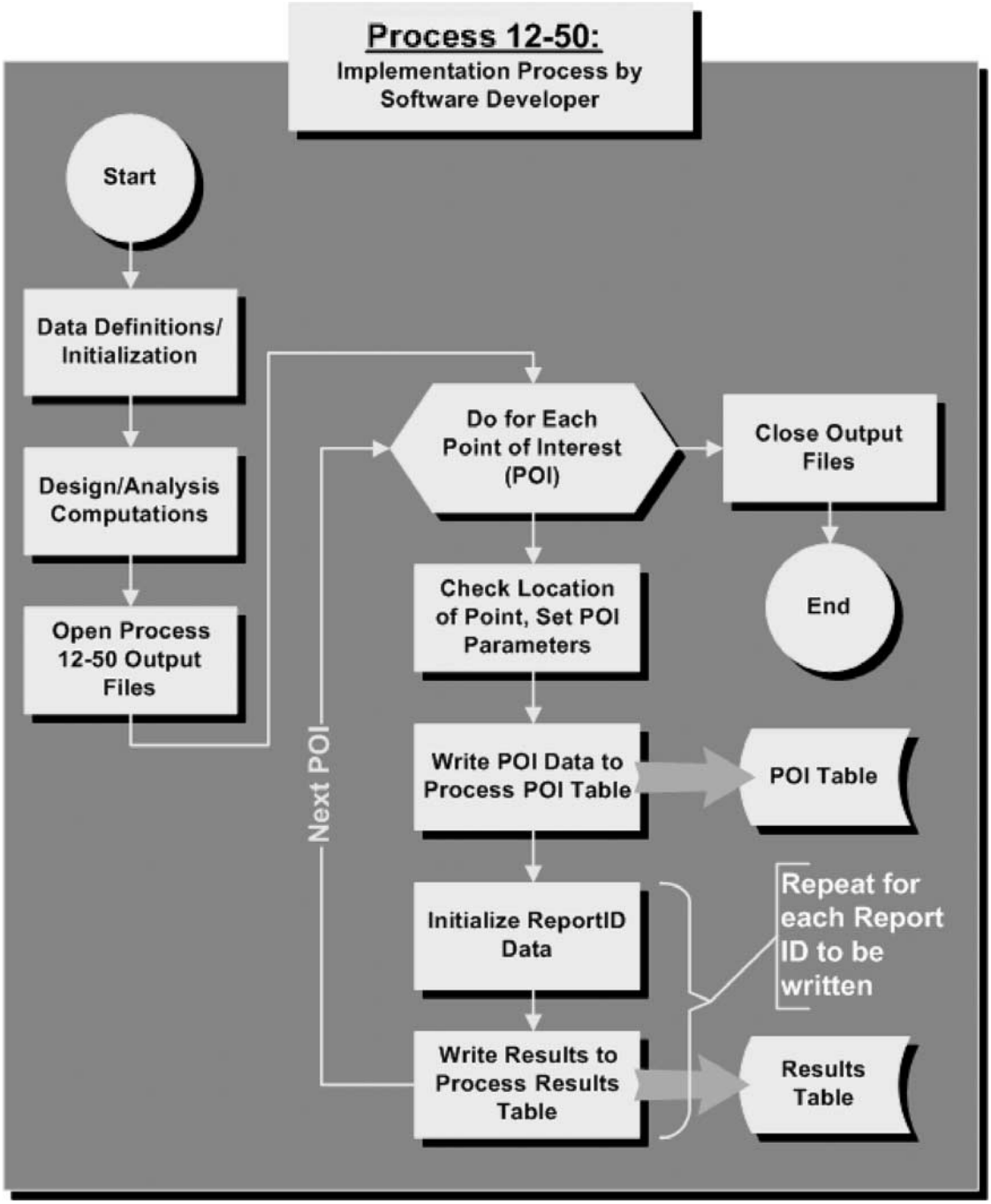

Figure F-4. Process 12-50 implementation flowchart.

12. Test the Process 12-50 Implementation

To effectively test the newly developed software, the Process 12-50 results must be written correctly. One of the best ways to test a Process 12-50 implementation is to set up a graphical viewing tool like the one available on the accompanying CD-ROMs. If one downloads the tools and some results produced by software from another developer many times, output differences will be relatively easy to spot.

13. Test the Newly Developed Software on the Web Site

Once one is fairly comfortable with the accuracy of the Process 12-50 implementation, there are several options to continue testing. First, other results from the accompanying CD-ROMs can be used and compared to those from the specific process. Use the tools available from the accompanying CD-ROMs to view results, modify the tools from these CD-ROMs to suit specific needs, or start from scratch and develop a viewer for comparing results. One area where this is particularly valuable is regression testing. One can store the results from different versions of the software and quickly compare them to catch any unexpected differences.

See the glossary for terms specific to Process 12-50.

VISUAL BASIC EXAMPLE

The following is a simple Microsoft Visual Basic example showing the implementation of Process 12-50.

Form Source Code

```
Option Explicit
Private Sub RunAnalysis_Click()
   Call Initialize
   Call RunTheAnalysis
   Call SaveNCHRPEngineData
   Call SavePOIData
   Call SaveNCHRPResultsTable
End Sub
```

Module 1 Source Code

```
Option Explicit

Dim sPath As String ‘//The save path

Public Sub Initialize()

   ‘//set up the AnalysisData structure//’

   sPath = App.Path ‘“c:\NCHRP\NCHRP-VB\Results\”

   AnalysisData.iProcessID = 2
   AnalysisData.sEngineName = “ExampleEngineVB”
   AnalysisData.sVersion = “1.00.00”
   AnalysisData.iNumberOfPOIs = 10
   AnalysisData.dUniformLoad = 1
   AnalysisData.dSpanLength = 100
   AnalysisData.iBridgeID = 1111
   AnalysisData.iSubDomainID = 53
   AnalysisData.iAuxID = 1

   POIData.iTenthPoint = 1
   POIData.iTwentiethPoint = 0

End Sub

Public Sub RunTheAnalysis()
   Dim i                As Integer      ‘counter
   Dim dRa              As Double       ‘left rection
   Dim dRb              As Double       ‘right reaction
   Dim dXIncrement      As Double       ‘increment distance between POIs

   ‘Compute Reactions
   dRa = AnalysisData.dUniformLoad * AnalysisData.dSpanLength / 2
   dRb = AnalysisData.dUniformLoad * AnalysisData.dSpanLength / 2

   ‘Compute x-location increment
   dXIncrement = AnalysisData.dSpanLength / 10

   ‘Compute shear and moment
   For i = 0 To 11
      aXLocation(i) = dXIncrement * i
      aVx(i) = dRa - AnalysisData.dUniformLoad * aXLocation(i)
      aMF(i) = dRa * aXLocation(i) - (0.5 * AnalysisData.dUniformLoad * aXLocation(i) *
aXLocation(i))
   Next i

End Sub
```

```
Public Sub SaveNCHRPEngineData()

    Dim sFileName As String
    sFileName = “EngineFile.txt”
    ChDir sPath

    Dim theDate
    Dim theMonth
    Dim theYear

    theDate = Date
    ‘theMonth = Month
    ‘theYear = Year

    Open (sPath + sFileName) For Output As #1
        Print #1, AnalysisData.iProcessID & “,”;
        Print #1, Chr$(34) & AnalysisData.sEngineName & Chr$(34) & “,”;
        Print #1, Chr$(34) & theDate & Chr$(34) & “,”;
        Print #1, Chr$(34) & AnalysisData.sVersion & Chr$(34)
    Close #1

End Sub

Public Sub SavePOIData()

    Dim i                 As Integer      ‘//Coutner
    Dim dPercentOfSpan    As Double
    Dim sFileName         As String

    sFileName = “POI.txt”
    ChDir sPath
    Open (sPath + sFileName) For Output As #1

    For i = 0 To AnalysisData.iNumberOfPOIs
        dPercentOfSpan = 100 * (aXLocation(i) / AnalysisData.dSpanLength)

        ‘These 3 are updated each loop
        If (i = 0 Or i = 10) Then
            POIData.iSS_BearingLoc = 1
            POIData.iExteriorSupport = 1
        Else
            POIData.iSS_BearingLoc = 0
            POIData.iExteriorSupport = 0
        End If

        If i = 5 Then
            POIData.iSS_MidSpan = 1
        Else
            POIData.iSS_MidSpan = 0
        End If

        Print #1, (i + 1) & “,”;
        Print #1, AnalysisData.iBridgeID & “,”;
        Print #1, AnalysisData.iProcessID & “,”;
        Print #1, AnalysisData.dSpanLength & “,”;
        Print #1, dPercentOfSpan & “,”;
        Print #1, AnalysisData.iSubDomainID & “,”;
        Print #1, POIData.iTwentiethPoint & “,”;
        Print #1, POIData.iTenthPoint & “,”;
        Print #1, POIData.iSS_BearingLoc & “,”;
        Print #1, POIData.iSS_MidSpan & “,”;
        Print #1, POIData.iPS_TransferLength & “,”;
        Print #1, POIData.iPS_DrapePoint & “,”;
        Print #1, POIData.iExteriorSupport & “,”;
        Print #1, POIData.iInteriorSupport & “,”;
```

```
        Print #1, POIData.iConcLoadLoc & “,”;
        Print #1, POIData.iTopFlangeCutoff & “,”;
        Print #1, POIData.iBottomFlangeCutoff & “,”;
        Print #1, POIData.iWebCutoff & “,”;
        Print #1, POIData.iDiaphragmLocation & “,”;
        Print #1, POIData.iUserDefined

    Next i

    Close #1

End Sub

Public Sub SaveNCHRPResultsTable()

    Dim i                As Integer      ‘Counter
    Dim sFileName        As String

    sFileName = “ResultsFile.txt”
    ChDir sPath
    Open (sPath + sFileName) For Output As #1

        For i = 0 To AnalysisData.iNumberOfPOIs

            ‘Do Shears First - report ID 30000
            Print #1, AnalysisData.iBridgeID & “,”;
            Print #1, AnalysisData.iProcessID & “,”;
            Print #1, 30000 & “,”;
            Print #1, AnalysisData.iReportID & “,”;
            Print #1, aXLocation(i) & “,”;
            Print #1, aVx(i) & “,”;
            Print #1, AnalysisData.iSubDomainID & “,”;
            Print #1, i & “,”;
            Print #1, AnalysisData.iAuxID

        Next i
        For i = 0 To AnalysisData.iNumberOfPOIs

            ‘Now Do Moments - report ID 30001
            Print #1, AnalysisData.iBridgeID & “,”;
            Print #1, AnalysisData.iProcessID & “,”;
            Print #1, 30001 & “,”;
            Print #1, AnalysisData.iReportID & “,”;
            Print #1, aXLocation(i) & “,”;
            Print #1, aMF(i) & “,”;
            Print #1, AnalysisData.iSubDomainID & “,”;
            Print #1, i & “,”;
            Print #1, AnalysisData.iAuxID

        Next i

    Close #1

End Sub
```

Module 2 Source Code

```
Option Explicit

Public POIData As New CPOIData
Public AnalysisData As New cAnalysisData

Public aXLocation(11)     As Double     ‘An array of locations
Public aMF(11)            As Double     ‘An array of moments
Public aVx(11)            As Double     ‘An array of shears
```

CAnalysisData Class

```
Option Explicit

Public sEngineName            As String       'Engine Name
Public sVersion               As String       'Version Number
Public sExecutionDate         As String       'Date that this analysis was executed
Public sFileMode              As String       'append or at start (rewind)

Public dUniformLoad           As Double       'A uniform load
Public dSpanLength            As Double       'The span length

Public iNumberOfPOIs          As Long         'The # of points of interest
Public dXValue                As Double       'Local X value
Public dYValue                As Double       'Local Y value
Public dNodeDistance          As Double       'Distance to POI

Public iSubDomainID           As Long         'Sub-domain ID (The type of computation)
Public iReportID              As Long         'Report ID (Assigned per process)
Public iBridgeID              As Long         'Bridge ID
Public iProcessID             As Long         'ProcessID
Public iPointID               As Long         'The Point of Interest ID
Public iLocationID            As Long         'Points to location table
Public iAuxID                 As Long         'The auxilary table ID
Public iLimitStateID          As Long         'Limit State ID
Public iMaxMin                As Double       'Max of min load effect
Public iNCHRPResultsUnit      As Double       'unit for results table
Public iNCHRPEngineUnit       As Double       'Unit for engine table
Public iPOIUnit               As Double       'Unit for POI table
```

CPOIDAT Class

```
Option Explicit

Public iSpanNum               As Long         'Span Number
Public dPrecentOfSpan         As Double       'Percent

''These are boolean values, but declared integer because
'' printing boolean values results in alpha (True/False) output
'' rather than 0 or 1.

Public iTwentiethPoint        As Integer      '1/20th Point flag
Public iTenthPoint            As Integer      '1/10th Point flag
Public iSS_BearingLoc         As Integer      'Bearing Location
Public iSS_MidSpan            As Integer      'Midspan
Public iPS_TransferLength     As Integer      'Transfer Length
Public iPS_DrapePoint         As Integer      'Drape Point
Public iExteriorSupport       As Integer      'Exterior Support
Public iInteriorSupport       As Integer      'Interior Support
Public iConcLoadLoc           As Integer      'is there a concentrated load
Public iTopFlangeCutoff       As Integer      'Location of top flange cut-off
Public iBottomFlangeCutoff    As Integer      'Location of bottom flange cut-off
Public iWebCutoff             As Integer      'location of web cut-off
Public iDiaphragmLocation     As Integer      'location of diaphragms
Public iUserDefined           As Integer      'user defined index
Public iPointID               As Integer      'primary key value
```

Results File

```
1111,2,30000,0,0,50,53,0, 1
1111,2,30000,0,10,40,53,1, 1
1111,2,30000,0,20,30,53,2, 1
1111,2,30000,0,30,20,53,3, 1
```

```
1111,2,30000,0,40,10,53,4, 1
1111,2,30000,0,50,0,53,5, 1
1111,2,30000,0,60,-10,53,6, 1
1111,2,30000,0,70,-20,53,7, 1
1111,2,30000,0,80,-30,53,8, 1
1111,2,30000,0,90,-40,53,9, 1
1111,2,30000,0,100,-50,53,10, 1
1111,2,30001,0,0,0,53,0, 1
1111,2,30001,0,10,450,53,1, 1
1111,2,30001,0,20,800,53,2, 1
1111,2,30001,0,30,1050,53,3, 1
1111,2,30001,0,40,1200,53,4, 1
1111,2,30001,0,50,1250,53,5, 1
1111,2,30001,0,60,1200,53,6, 1
1111,2,30001,0,70,1050,53,7, 1
1111,2,30001,0,80,800,53,8, 1
1111,2,30001,0,90,450,53,9, 1
1111,2,30001,0,100,0,53,10, 1
```

FORTRAN EXAMPLE

NCHRP Fortran Example Main.f90

```
    Program NCHRPFortranExampleMain

    implicit none

    character *20 cEngineName           !Engine name
    character *8 cVersion               !Version no.
    character(9) cToday                 !Execution Date
    character *6 cFileMode              !append or at start (rewind)

    real *8 w                           ! uniform load
    real *8 span                        ! span length
    real *8 Xloc(11)                    ! locations
    real *8 MF(11)                      ! moments
    real *8 Vx(11)                      ! shears
    integer *4 NoOfPOIs                 ! number of points of interest (POIs)

    real *8 fXValue                     !local x value
    real *8 fYValue                     !local y value
    real *8 fNodeDistance               !distance to POI
    real *8 fSpanLength                 !span length

    integer *4 iSubdomainID         !subdomain ID, what kind of computation is this
    integer *4 iEngineID            !engine ID, assigned to developer
    integer *4 iReportID            !report ID, assigned per process and what is written
    integer *4 iBridgeID            !bridge ID
    integer *4 iProcessID           !processID
    integer *4 iPointId             !POI ID
    integer *4 iLocationID          !location ID, points to location table
    integer *4 iAuxID               !auxilary table ID
    integer *4 iNchrpResultsUnit    !unit for results table
    integer *4 iNchrpEngineUnit     !unit for engine table
    integer *4 iPOI                 !unit for POI table
    integer *4 i                    !loop counter

! POI definition =1 true, 0 = false

    integer *4 iSpanNum             !span number
    integer *4 fPercentOfSpan       !percent

    integer *4 iTwentiethPoint      !1/20 pt?
    integer *4 iTenthPoint          !1/10 pt
```

```
   integer *4 iSS_BearingLoc       !bearing location?
   integer *4 iSS_Midspan          !at midspan?
   integer *4 iPS_TransferLength   !end of transfer length?
   integer *4 iPS_DrapePoint       !at drape point?
   integer *4 iExteriorSupport     !is this an exterior support?
   integer *4 iInteriorSupport     !is this an interior support?
   integer *4 iConcLoadLoc         !is there a concentrated load?
   integer *4 iTopFlangeCutoff     !location of flange cut off?
   integer *4 iBotFlangeCutoff     !location of flange cut off?
   integer *4 iWebCutoff           !location of web cut off?
   integer *4 iDiaphragmLoc        !location of diaphragm?
   integer *4 iUserDefined         !user-defined index.

! end POI definition tags

   logical check                   !just return from routine
   logical SimpleAnalysis          !computational routine
   logical WriteNCHRP              !boolean to execute or skip writing NCHRP data
   logical WriteNCHRPResultsTable  !logical function
   logical WriteNCHRPPoiTable      !logical function

   data iEngineId       / 2/                !engine ID
   data iProcessId      / 2/           !process ID
   data iSubdomainID        /53/   !a subdomain, 53 is an example only, it doesn't
                                   !currently exist
   data cEngineName     /'ExampleEngine'/    !Engine name
   data cVersion        /'1.00.00'/          !version number
   data WriteNCHRP      /.true. /            !yes, write the data

!========================================= Computational portion of this code
===========================================

! We will assume that this computation provides the shear and moments for the self wt of a
girder.
! The NCHRP report IDs are xxx and yyy, respectively.

   NoOfPOIs  = 11                  !number of POIs
   w         = 1.                  !set uniform load
   span      = 100.                !set span length
   iBridgeID = 1111                !set bridge ID

      !call analysis, computes Vx and MF

   check = SimpleAnalysis (w, span, xloc, Vx, MF)

!========================================= NCHRP process portion of this code

! Set up file necessary for output tables ++++++++++++++++

   iNchrpResultsUnit = 1           !assign file unit number for results table
   iNchrpEngineUnit  = 2           !assign file unit number for engine table
   iPOI              = 3           !assign file unit number for POI table

!     cFileMode = 'APPEND'                                !set your choice
   cFileMode = 'REWIND'

   Open (iNchrpResultsUnit,file='ResultsFile.txt',position=cFileMode,status = 'unknown')

   Open (iNchrpEngineUnit ,file='EngineFile.txt' ,position = cFileMode, status = 'unknown')

   Call Date(cToday)                                           !get date

! Write engine data once, this does not need to be written every execution and can be added
manually if desired.

   write(iNchrpEngineUnit,300) iEngineID, cEngineName, cToday, cVersion

300 format(i2,                    &        !EngineID
           ',' , '"',A20, '"',    &        !Engine
```

```
            ',' , '"',A9 , '"',   &        !Release/Test Date Associate with Version
            ',' , '"',A8 , '")             !Version

   close (iNchrpEngineUnit)

! End engine data table write.+++++++++++++++++++++++++

! Write a table that outlines what the POI are used for, i.e., why are they POI.
! Typically this is set up once in a program/process

   Open (iPOI ,file='POI.txt' ,position = cFileMode, status = 'unknown')

   fSpanLength = span              !assign span length

   iSpanNum = 1

   do i = 1, NoOfPOIs                   !loop for each POI

      fNodeDistance = xloc(i)      !assign location
      iPointId = i                 !assign primary key

      iTwentiethPoint = 0
      iTenthPoint     = 1

      if (i == 1 .or. i == NoOfPOIs) then
         iSS_BearingLoc   = 1
         iExteriorSupport = 1
      else
         iSS_BearingLoc   = 0
         iExteriorSupport = 0
      end if

      if (i == 6) then
         iSS_Midspan = 1
      else
         iSS_Midspan = 0
      end if

      iPS_TransferLength = 0
      iPS_DrapePoint     = 0

      iConcLoadLoc       = 0

      iTopFlangeCutoff   = 0
      iBotFlangeCutoff   = 0
      iWebCutoff         = 0

      iDiaphragmLoc      = 0
      iUserDefined       = 0

! pass data to write, this is usually performed once for each POI
! during a program/process

      check =              &     !call write function

              WriteNCHRPPoiTable( &

                     WriteNCHRP,       &      !see function for definitions =1,
                                              ! true/yes =0, false/no
                     iPOI,                    &
                     iBridgeID,               &
                     iProcessId,              &
                     iPointId,                &
                     iSpanNum,                &
                     iSubdomainID,            &
                     iTwentiethPoint,         &
                     iTenthPoint,             &
                     iSS_BearingLoc,          &
                     iSS_Midspan,             &
```

```
                    iPS_TransferLength,      &
                    iPS_DrapePoint,          &
                    iExteriorSupport,        &
                    iInteriorSupport,        &
                    iConcLoadLoc,            &
                    iTopFlangeCutoff,        &
                    iBotFlangeCutoff,        &
                    iWebCutoff,              &
                    iDiaphragmLoc,           &
                    iUserDefined,            &
                    fNodeDistance,           &
                    fSpanLength              )

      end do
! End writing POI Table ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
! Write data associated with results/computations
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

! This is performed many times during a program for each reportID available

! Set up parameters for writing data. This is usually necessary for each report ID.

   iAuxID = 1      !Foreign key to auxilary table (could be used for numerous
                   !reasons), see xxx.xxx
                   !For this example, this will always be 1

! Write data to the results table. Local variables are used here, but this can be
! accomplish in a host of ways.

   do iLocationID = 1, NoOfPOIs         !loop for each POI

      fXValue = xloc(iLocationID)             !location
      fYValue = Vx(iLocationID)               !shear
      iReportID = 30001                       !look this up

      check =                              &
         WriteNCHRPResultsTable(         &
         WriteNCHRP,            &
         iNchrpResultsUnit,     &
         iBridgeID,             &
         iReportID,             &
         iLocationID,           &
         iAuxID,                &
         iProcessID,            &
         iSubdomainID,          &
         fXValue,               &
         fYValue)

      fXValue = xloc(i)                 !location
      fYValue = MF(i)                   !moment
      iReportID = 30000                 !look this up

      check =                 &
         WriteNCHRPResultsTable(         &
            WriteNCHRP,                              &
            iNchrpResultsUnit,                       &
            iBridgeID,                               &
            iReportID,                               &
            iLocationID,                             &
            iAuxID,                                  &
            iProcessID,                              &
            iSubdomainID,                            &
            fXValue,                                 &
            fYValue)
```

```
   end do                                                    !close POI loop

   Close(iNchrpResultsUnit)                   !results table
   Close(iPOI)                                               !POI table

   stop
   end
```

SimpleAnalysis.f90

```
   Function SimpleAnalysis (w, span, xloc, Vx, MF)

   implicit none

      real *8 Span          ! span length

      real *8 w             ! Uniform load

      real *8 Ra            ! left reaction
      real *8 Rb            ! right reaction

      real *8 Xloc(11)      ! locations
      real *8 MF(11)        ! moments
      real *8 Vx(11)        ! shears
      real *8 xinc          ! distance between points of interest

      integer *4 i          ! location loop index
      integer *4 EngineId   ! Engine id code

      logical SimpleAnalysis

C  Compute Reactions
      Ra = w * Span/2.
      Rb = w * Span/2.

C  Compute xlocation increment
      Xinc = span / 10.0

C  Compute shear and moments
   Do i = 1, 11
            Xloc(i) = Xinc * float( (i-1) )
            Vx(i) = Ra - (w * Xloc(i))
            MF(i) = Ra * Xloc(i) - 0.5*( w * Xloc(i)**2 )

   End Do

      SimpleAnalysis = .true.

   Return
      End
```

WriteNCHRPPOITable.f90

```
Function WriteNCHRPPoiTable(     &
          WriteNCHRP,            &
                     iPOI,                &
          iBridgeID,             &
          iProcessId,            &
                     iPointId,            &
                     iSpanNum,            &
          iSubDomainId,          &
          iTwentiethPoint,       &
          iTenthPoint,           &
          iSS_BearingLoc,        &
          iSS_Midspan,           &
          iPS_TransferLength,    &
          iPS_DrapePoint,        &
          iExteriorSupport,      &
```

```
      iInteriorSupport,      &
      iConcLoadLoc,          &
      iTopFlangeCutoff,      &
      iBotFlangeCutoff,      &
      iWebCutoff,            &
      iDiaphragmLoc,         &
      iUserDefined,          &
      fNodeDistance,         &
      fSpanLength            )

   implicit none

! Argument list variables.

      logical WriteNCHRP
      integer *4 iPOI
      integer *4 iNCHRP_PointId
      integer *4 iBridgeID
      integer *4 iProcessId
      integer *4 iPointId
      integer *4 iSpanNum
      integer *4 iSubDomainId
      integer *4 iTwentiethPoint
      integer *4 iTenthPoint
      integer *4 iSS_BearingLoc
      integer *4 iSS_Midspan
      integer *4 iPS_TransferLength
      integer *4 iPS_DrapePoint
      integer *4 iExteriorSupport
      integer *4 iInteriorSupport
      integer *4 iConcLoadLoc
      integer *4 iTopFlangeCutoff
      integer *4 iBotFlangeCutoff
      integer *4 iWebCutoff
      integer *4 iDiaphragmLoc
      integer *4 iUserDefined
      real    *8 fPercentOfSpan

      ! Local variables

      logical WriteNCHRPPoiTable      !logical function
      real *8 fNodeDistance           !absolute distance where POI is
                                              ! located
      real *8 fSpanLength             !span length

      ! Determine if the record should be written.
      if (.NOT. WriteNCHRP) return

      ! Compute the percent of span.
      fPercentOfSpan = 100.0 * fNodeDistance / fSpanLength

      ! Write record to POI file.
      write(iPOI,1000)                &
            iPointId,               &
            IBridgeID,              &
            iProcessId,             &
            iSpanNum,               &
            fPercentOfSpan,         &
            iSubDomainId,           &
            iTwentiethPoint,        &
            iTenthPoint,            &
            iSS_BearingLoc,         &
            iSS_Midspan,            &
            iPS_TransferLength,     &
```

```
                iPS_DrapePoint,         &
                iExteriorSupport,       &
                iInteriorSupport,       &
                iConcLoadLoc,           &
                iTopFlangeCutoff,       &
                iBotFlangeCutoff,       &
                iWebCutoff,             &
                iDiaphragmLoc,          &
                iUserDefined
          1000 format(i5,',',i4,',',i4,',',i4,',',f10.4,',',i4,14(',',i3))

          WriteNCHRPPoiTable = .true.

return
End
```

WriteNCHRPResultsTable.f90

```
function WriteNCHRPResultsTable(WriteNCHRP, iNchrpResultsUnit, &
                        iBridgeID, iReportID, iLocationID, &
                    iAuxID,iProcessID, iSubdomainID, fXValue, fYValue)
! LocationID: Identification number for a location along the girderline
 fXValue:    Actual position along the girderline

       implicit none

       integer *4 iBridgeID   ! bridge ID
       integer *4 iProcessID  ! assign process ID
       integer *4 iSubdomainID        ! Subdomain ID
       integer *4 iReportID   ! report ID
       integer *4 iLocationID ! location ID
       integer *4 iAuxId              ! NCHRP load ID
       integer *4 ilsID               ! limit state ID
       integer *4 iMaxMin     ! indicator for maximum & minimum effects
       integer *4 iNchrpResultsUnit   ! unit for results table

       real    *8 fXValue, fYValue    ! x and y values

       logical WriteNCHRP
       logical WriteNCHRPResultsTable

       ! Determine if the record should be written.
       if (.NOT. WriteNCHRP) return

       write(iNchrpResultsUnit,1000)    &
                                            iBridgeID,    &
                                            iProcessID,   &
                                            iReportID,    &
                                            fXValue,      &
                                            fYValue,      &
                                            iSubdomainID, &
                                            iLocationID,  &
                                            iAuxId

1000   format(i4,',',i4,',',i6,2(',', &
              G15.7),',',i4,',',i5,',',i3,',',i2,',',i1)

       WriteNCHRPResultsTable = .true.

       return
End
```

VISUAL C++ EXAMPLE

NCHRP C++ Example Main.cpp

```
//
//     Written by Mark C Jablin
//     BridgeTech, Inc.
//     9/9/2001
//
//     Abstract:    Example code for NCHRP 12-50 process in C++.
//                          Translated from Jay Puckett's Fortran example.
//                          This is the main program
//
//
//     Modifications:
//
//
//
//

#include "SimpleAnalysis.h"
#include "POIData.h"
#include <iostream>

int main()
{
       try
       {
              SimpleAnalysis NCHRPAnalysis; //create an instance of the analysis object
              NCHRPAnalysis.RunAnalysis();              //perform the analysis
              NCHRPAnalysis.WriteNCHRPEngineData();

              //write POI information
              for(long i=0;i<NCHRPAnalysis.GetNoOfPOIs();i++)
              {
                     POIData data(i+1,   //generate the POI data, tenth point
                                  1,    //span number
                                 NCHRPAnalysis.GetNoOfPOIs());//number of POIs
                     NCHRPAnalysis.WriteNCHRPPoiTable(data);
              }

              long iAuxID = 1;           //Foreign key to auxilary table
                                         //(could be used for numerous reasons),
                                         //see xxx.xxx

              // Write data to the results table. Local variables are used here, but
              // this can be accomplish in a host of ways.

              for(i=0;i<NCHRPAnalysis.GetNoOfPOIs();i++)//loop for each POI
              {

                     NCHRPAnalysis.SetReportID(30001);
                     NCHRPAnalysis.WriteNCHRPResultsTable(NCHRPAnalysis.GetMoments(),
                                                                  i);
                     NCHRPAnalysis.SetReportID(30000);
                     NCHRPAnalysis.WriteNCHRPResultsTable(NCHRPAnalysis.GetShears(),
                                                                  i);
              }
       }
       catch(std::exception &e)
       {
              std::cout << e.what() << std::endl;
              return 0;
```

```
	}
	catch(...)
	{
		std::cout << “Unknown exception, terminating execution.” << std::endl;
		return 0;
	}
	return 1;
}
```

POIData.cpp

```
//
//	Written by Mark C Jablin
//	BridgeTech, Inc.
//	9/10/2001
//
//	Abstract: Implementation for the class that stores all the data required for the
//		  NCHRP 12-50 POI table.
//
//
//
//
//	Modifications:
//
//
//
//

#include “POIData.h”

POIData::POIData(long iTenthPoint, long iSpanNumber, long iNoOfPOIs):
	m_bPS_TransferLength(false),
	m_bPS_DrapePoint(false),
	m_bConcLoadLoc(false),
	m_bTopFlangeCutoff(false),
	m_bBotFlangeCutoff(false),
	m_bWebCutoff(false),
	m_bDiaphragmLoc(false),
	m_bUserDefined(false),
	m_bInteriorSupport(false),
	m_bTwentiethPoint(false),
	m_bTenthPoint(true),
	m_iSpanNum(iSpanNumber),
	m_iPointID(iTenthPoint)
{
	if (iTenthPoint == 1 || iTenthPoint == iNoOfPOIs)
	{
	   m_bSS_BearingLoc   = true;
	   m_bExteriorSupport = true;
	}
	else
	{
	   m_bSS_BearingLoc   = false;
	   m_bExteriorSupport = false;
	}
	if (iTenthPoint == 6)
	{
	   m_bSS_Midspan = true;
	}
	else
	{
	   m_bSS_Midspan = false;
```

```
        }
}
long    POIData::GetSpanNumb()
{
        return m_iSpanNum;
}
double POIData::GetPercentOfSpan()
{
        return m_dPercentOfSpan;
}
bool    POIData::GetTwentiethPoint()
{
        return m_bTwentiethPoint;
}
bool    POIData::GetTenthPoint()
{
        return m_bTenthPoint;
}
bool    POIData::GetSS_BearingLoc()
{
        return m_bSS_BearingLoc;
}
bool    POIData::GetSS_Midspan()
{
        return m_bSS_Midspan;
}
bool    POIData::GetPS_TransferLength()
{
        return m_bPS_TransferLength;
}
bool    POIData::GetPS_DrapePoint()
{
        return m_bPS_DrapePoint;
}
bool    POIData::GetExteriorSupport()
{
        return m_bExteriorSupport;
}
bool    POIData::GetInteriorSupport()
{
        return m_bInteriorSupport;
}
bool    POIData::GetConcLoadLoc()
{
        return m_bConcLoadLoc;
}
bool    POIData::GetTopFlangeCutoff()
{
        return m_bTopFlangeCutoff;
}
bool    POIData::GetBotFlangeCutoff()
{
        return m_bBotFlangeCutoff;
```

```
}

bool   POIData::GetWebCutoff()
{
       return m_bWebCutoff;
}

bool   POIData::GetDiaphragmLoc()
{
       return m_bDiaphragmLoc;
}

bool   POIData::GetUserDefined()
{
       return m_bUserDefined;
}

long   POIData::GetPointID()
{
       return m_iPointID;
}
```

SimpleAnalysis.cpp

```
//
//     Written by Mark C Jablin
//     BridgeTech, Inc.
//     9/9/2001
//
//     Abstract: Example code for NCHRP 12-50 process in C++
//               Translated from Jay Puckett's Fortran example
//
//
//
//     Modifications:
//
//
//
//

#include "SimpleAnalysis.h"

SimpleAnalysis::SimpleAnalysis() throw (std::exception)
               :    m_iEngineID(2),          //data initialization
                    m_iProcessID(2),
                    m_sEngineName("ExampleEngineCPP1"),
                    m_sVersion("1.00.00"),
                    m_bWriteNCHRP(true),     //this will be a input parameter
                                             //usually, and not set to true
                                             //all the time
                    m_iNoOfPOIs(11),
                    m_dw(1.0),
                    m_dSpan(100.0),
                    m_iBridgeID(1111),
                    m_dXloc(11),
                    m_dMF(11),
                    m_dVx(11),
                    m_iSubDomainID(53),
                    m_iAuxID(1)
{
       m_fNCHRPOutput.open("ResultsFile.txt");
       m_fEngineOutput.open("EngineFile.txt");
       m_fPOIOutput.open("POI.txt");
```

```
        if(!m_fNCHRPOutput || !m_fEngineOutput || !m_fPOIOutput)
        {
          std::exception e("Couldn't open all the files. Simple Analysis constructor");
          throw e;
        }
}

void SimpleAnalysis::WriteNCHRPPoiTable(POIData data) throw (std::exception)
                              //writes POI table

{
        try
        {
                double dPercentOfSpan;                  //percent of the span length

                // Determine if the record should be written.

                if (!m_bWriteNCHRP) return;

                // Compute the percent of span.
                dPercentOfSpan = 100.0 * m_dXloc[data.GetPointID() - 1] / m_dSpan;

                m_fPOIOutput << data.GetPointID()             << ",";
                m_fPOIOutput << m_iBridgeID                   << ",";
                m_fPOIOutput << m_iProcessID                  << ",";
                m_fPOIOutput << m_dSpan                       << ",";
                m_fPOIOutput << dPercentOfSpan                << ",";
                m_fPOIOutput << m_iSubDomainID                << ",";
                m_fPOIOutput << data.GetTwentiethPoint()      << ",";
                m_fPOIOutput << data.GetTenthPoint()          << ",";
                m_fPOIOutput << data.GetSS_BearingLoc()       << ",";
                m_fPOIOutput << data.GetSS_Midspan()          << ",";
                m_fPOIOutput << data.GetPS_TransferLength()   << ",";
                m_fPOIOutput << data.GetPS_DrapePoint()       << ",";
                m_fPOIOutput << data.GetExteriorSupport()     << ",";
                m_fPOIOutput << data.GetInteriorSupport()     << ",";
                m_fPOIOutput << data.GetConcLoadLoc()         << ",";
                m_fPOIOutput << data.GetTopFlangeCutoff()     << ",";
                m_fPOIOutput << data.GetBotFlangeCutoff()     << ",";
                m_fPOIOutput << data.GetWebCutoff()           << ",";
                m_fPOIOutput << data.GetDiaphragmLoc()        << ",";
                m_fPOIOutput << data.GetUserDefined()         << std::endl;
        }
        catch(...)
        {
                std::exception e("Unknown exception writing to POI table.");
                throw e;
        }
}

void SimpleAnalysis::WriteNCHRPResultsTable(std::vector<double> dYValue,
                                            long iIndex) throw (std::exception)
                        //writes results table
{
        try
        {
                //Determine if the record should be written.
                if (!m_bWriteNCHRP) false;
                m_fNCHRPOutput << m_iBridgeID                 << ",";
                m_fNCHRPOutput << m_iProcessID                << ",";
                m_fNCHRPOutput << m_iReportID                 << ",";
                m_fNCHRPOutput << m_dXloc[iIndex]             << ",";
                m_fNCHRPOutput << dYValue[iIndex]             << ",";
```

```
            m_fNCHRPOutput << m_iSubDomainID              << “,”;
            m_fNCHRPOutput << iIndex                      << “,”;
            m_fNCHRPOutput << m_iAuxID                    << std::endl;
      }
      catch(...)
      {
            std::exception e(“Unknown exception writing to Results table.”);
            throw e;
      }
}

void SimpleAnalysis::WriteNCHRPEngineData() throw (std::exception)//writes engine data table
{
      try
      {
            //get the time
            time_t now = time(0);
            tm CurrentTime = *localtime(&now);

            long iHour, iMinute, iSecond, iDay, iMonth, iYear;

            iHour = CurrentTime.tm_hour;
            iMinute = CurrentTime.tm_min;
            iSecond = CurrentTime.tm_sec;
            iDay = CurrentTime.tm_mday;
            iMonth = CurrentTime.tm_mon + 1; //months are 0 to 11, convert
            iYear = CurrentTime.tm_year + 1900;//00 is 1900

            //write the first field
            m_fEngineOutput << m_iEngineID;            //write the engine ID
            m_fEngineOutput << “,”;                    //write a comma
            m_fEngineOutput << ‘“’;                    //open text
            m_fEngineOutput << m_sEngineName;          //write the engine name
            m_fEngineOutput << ‘“’;                    //close text

            //repeat for each additional field
            m_fEngineOutput << “,”;                    //write a comma

            m_fEngineOutput << ‘“’;
            if(iMonth < 10)
               m_fEngineOutput << “0”;                 //add a zero the month is less than
                                                       // 10 (nicer format)
            m_fEngineOutput << iMonth;
            m_fEngineOutput << “-”;
            if(iDay <10)
               m_fEngineOutput << “0”;                 //add a Zero the day is less than 10
                                                       //(nicer format)
            m_fEngineOutput << iDay;
            m_fEngineOutput << “-”;
            m_fEngineOutput << iYear;
            m_fEngineOutput << ‘“’;

            m_fEngineOutput << “,”;

            m_fEngineOutput << ‘“’;
            m_fEngineOutput << m_sVersion;
            m_fEngineOutput << ‘“’<< std::endl;
      }
      catch(...)
      {
            std::exception e(“Unknown exception writing to POI table.”);
            throw e;
      }
}
```

```
void SimpleAnalysis::RunAnalysis() throw (std::exception)
                                                //performs analysis
{
	try
	{
		double dRa;		// left reaction
		double dRb;		// right reaction

		double dXinc;		// distance between points of interest

		long i;		// location loop index

		//Compute Reactions
		dRa = m_dw * m_dSpan/2.0;
		dRb = m_dw * m_dSpan/2.0;

		//Compute xlocation increment
		dXinc = m_dSpan / 10.0;

		//Compute shear and moments
		for(i = 0;i<11;i++)
		{
			m_dXloc[i] = dXinc * static_cast<double>(i);
			m_dVx[i] = dRa - (m_dw * m_dXloc[i]);
			m_dMF[i] = dRa * m_dXloc[i] - 0.5*( m_dw * m_dXloc[i] *
								m_dXloc[i] );
		}
	}
	catch(...)
	{
		std::exception e("Unknown error running analysis.");
		throw e;
	}
}

long SimpleAnalysis::GetNoOfPOIs()
{
	return m_iNoOfPOIs;
}

void SimpleAnalysis::SetReportID(long iReportID)
{
	m_iReportID = iReportID;
}

std::vector<double> SimpleAnalysis::GetMoments()
{
	return m_dMF;
}
std::vector<double> SimpleAnalysis::GetShears()
{
	return m_dVx;
}
```

POIData.h

```
//
//	Written by Mark C Jablin
//	BridgeTech, Inc.
//	9/10/2001
//
//	Abstract: Header for the class that stores all the data required for the NCHRP
//		12-50 POI table.
//
```

```
//
//
//
//     Modifications:
//
//
//
//

#ifndef POIDATA_H
#define POIDATA_H

class POIData
{
public:
       POIData(long iTenthPoint, long iSpanNumber, long iNoPOIs);

       long   GetSpanNumb();
       double GetPercentOfSpan();
       bool   GetTwentiethPoint();
       bool   GetTenthPoint();
       bool   GetSS_BearingLoc();
       bool   GetSS_Midspan();
       bool   GetPS_TransferLength();
       bool   GetPS_DrapePoint();
       bool   GetExteriorSupport();
       bool   GetInteriorSupport();
       bool   GetConcLoadLoc();
       bool   GetTopFlangeCutoff();
       bool   GetBotFlangeCutoff();
       bool   GetWebCutoff();
       bool   GetDiaphragmLoc();
       bool   GetUserDefined();
       long   GetPointID();

private:

// POI definition =1 true, 0 = false
   long       m_iSpanNum;                          //span number
   double     m_dPercentOfSpan;               //percent
   bool       m_bTwentiethPoint;              //1/20 pt
   bool       m_bTenthPoint;                  //1/10 pt
   bool       m_bSS_BearingLoc;               //bearing location
   bool       m_bSS_Midspan;                  //at midspan
   bool       m_bPS_TransferLength;      //end of transfer length
   bool       m_bPS_DrapePoint;               //at drape point
   bool       m_bExteriorSupport;             //is this an exterior support
   bool       m_bInteriorSupport;             //is this an interior support
   bool       m_bConcLoadLoc;                       //is there a concentrated load
   bool       m_bTopFlangeCutoff;             //location of flange cut off
   bool       m_bBotFlangeCutoff;             //location of flange cut off
   bool       m_bWebCutoff;                   //location of web cut off
   bool       m_bDiaphragmLoc;                //location of diaphragm
   bool       m_bUserDefined;                       //user-defined index.
     long      m_iPointID;                          //primary key value
// end POI definition tags
};

#endif
```

SimpleAnalysis.h

```
//
//     Written by Mark C Jablin
```

```
//     BridgeTech, Inc.
//     9/9/2001
//
//     Abstract: Header file for C++ example program for NCHRP 12-50 process.
//                  Translated from Jay Puckett's Fortran example code.
//               This is the analysis class that contains all the data and
//
//
//     Modifications:
//
//
//
//
#ifndef SIMPLEANALYSIS_H
#define SIMPLEANALYSIS_H

#include <string>
#include <fstream>
#include <ctime>
#include <vector>
#include <exception>
#include "POIData.h"

class SimpleAnalysis
{

public:
       SimpleAnalysis() throw (exception);//constructor
       void WriteNCHRPResultsTable(std::vector<double> dYValue,
                                                          long iIndex);
       void WriteNCHRPPoiTable(POIData data) throw (std::exception);
       void WriteNCHRPEngineData() throw (std::exception);
       void RunAnalysis() throw (std::exception);
       //computes Vx and MF
       long GetNoOfPOIs();

       void SetReportID(long iReportID);
       std::vector<double> GetMoments();
       std::vector<double> GetShears();

private:
       std::string    m_sEngineName;         //Engine name
   std::string       m_sVersion;                    //Version no.
   std::string       m_sToday;                      //Execution Date
       std::string    m_sFileMode;           //append or at start (rewind)

       double  m_dw;                                 // uniform load
   double     m_dSpan;                               // span length
       std::vector<double> m_dXloc;     // locations (11)
       std::vector<double> m_dMF;            // moments (11)
       std::vector<double> m_dVx;            // shears (11)
       long    m_iNoOfPOIs;                  // number of points of interest (POIs)

       double  m_dXValue;                            //local x value
       double  m_dYValue;                            //local y value
   double     m_dNodeDistance;              //distance to POI
   double     m_dSpanLength;                //span length

   long       m_iSubDomainID;                       //subdomain ID, what kind of
computation is this
       long    m_iEngineID;                  //engine ID, assigned to developer
       long    m_iReportID;                  //report ID, assigned per process and
what is written
       long    m_iBridgeID;                  //bridge ID
```

```
   long       m_iProcessID;                  //processID
      long    m_iPointID;                            //POI ID
   long       m_iLocationID;                 //location ID, points to location table
   long       m_iAuxID;                              //auxilary table ID
      long    m_iLSID;                               //limit state ID
      long    m_iMaxMin;                             //max of min load effect
      long    m_iNchrpResultsUnit;     //unit for results table
      long    m_iNchrpEngineUnit;            //unit for engine table
      long    m_iPOI;                                //unit for POI table

   bool       m_bWriteNCHRP;                 //boolean to execute or skip writing NCHRP data
                                                                  //this would be an input
parameter in most codes

      std::ofstream m_fStandardOutput;
      std::ofstream m_fNCHRPOutput;
      std::ofstream m_fEngineOutput;
      std::ofstream m_fPOIOutput;
};

#endif
```

APPENDIX G
DATA ORGANIZATION

INTRODUCTION

Because of the large number of electronic files produced as a result of this research, it is necessary to describe the organization of these files to facilitate the use of the information. This appendix describes the data organization for Process 12-50.

Although the file organization may not be the most efficient, it should be intuitive enough for other users to find what they are looking for. The caretakers of the Process 12-50 data should review the file naming conventions/organization for improvements.

The data, which are provided on the two accompanying CD-ROMs in a compressed format, consist of the following:

- Raw data from each of the subdomains,
- Microsoft Excel Spreadsheets used to generate the input files for the subdomains,
- Microsoft Access databases with the results information for each subdomain,
- Microsoft Access database containing the Process 12-50 common tables,
- Common Process 12-50 tables in XML format,
- Utility software developed during the course of the project, and
- A viewer to aid in the graphical viewing of the Process 12-50 Microsoft Access databases.

The files and their organization are described in detail in the following sections.

FILE ORGANIZATION AND CD CONTENTS

To help users find information about a specific bridge, process, or subdomain, the files produced for Process 12-50 have been organized according to the criteria described in the following sections. The main folders of the two CD-ROMs are as shown in Figure G-1. [Note: These CD-ROMs are labeled CRP-CD-ROM 29A and CRP-CD-ROM 29B. In this appendix, CD-ROM 29A is referred to as CD1 and CD-ROM 29B is referred to as CD2.] Each of these folders is described in detail in the following sections.

‘Common Tables’ Folder (CD 1 and 2)

This folder contains information related to the common tables used by Process 12-50. The data for these tables are provided in both XML format (along with the associated document type definition (DTD) and schema files) and Microsoft Access format. The tables include the following:

- ProcessID
- ReportID
- SpecArticles
- SpecVersion
- SubReport
- Units

These tables are described in detail in Appendixes N and CC of the research team’s final report.

‘Spreadsheets’ Folder (CD 1 and 2)

During the development of Process 12-50, spreadsheets played an important role in the generation of the numerous input files required to test each subdomain. The use of the spreadsheets and how they fit in Process 12-50 is documented in Appendix A. The spreadsheets are provided “as is” and do not contain the official subdomain information. This information is provided in the ‘SubdomainRawData’ folder. The spreadsheets are provided as an example for automating the generation of a numerous input files for multiple processes.

The spreadsheets are contained in a compressed zipped file (‘Spreadsheets.zip’) and are named according to the subdomains they represent.

‘Software’ Folder (CD 2 and 2)

This folder contains programs created during the development of Process 12-50 for various processes, including testing the feasibility of using the XML format for Process 12-50 data, and conversion utilities. Also included is an Executable to view the Process 12-50 data contained in the Microsoft Access databases. The programs are delivered “as is.”

‘CommonTableXMLGenerator’ Folder

This folder contains a program written in Visual Basic and includes the source code along with an executable provided in the ‘Package’ subfolder.

The utility program converts the common tables for Process 12-50 Microsoft Access format to the XML format described in Appendix CC of the research team’s final report. The input for the program is the database *CommonTables.mdb* (found on CD 1 of 2 in the CommonTables folder) and the outputs are the various common tables in XML format (see Figure G-2).

Figure G-1. Main directory.

This program can be used to regenerate the XML files anytime the *CommonTables.mdb* database is changed.

'NCHRPXML Viewer' Folder

This folder contains the software used to test the feasibility and advantages of using XML as a Process 12-50 file format. The program uses an early version of the XML format developed for Process 12-50 and, therefore, does not work with any of the final XML data files. It does provide an example of using XML data with a programming language. Samples using the early version of the Process 12-50 are included in this folder to demonstrate the program.

'NCHRP Data Viewer' Folders

This folder contains an executable for installing a simple application for viewing the results of the Process 12-50 databases. To view Process 12-50 data, perform the following steps:

- Install the program by double clicking on \Software\NCHRP Data Viewer\SETUP.EXE. (CD 2 of 2)
- Install the subdomains you want to review from the following zip file \SubdomainAccessData\AllViewerData.zip (CD 2 of 2)
- From the 'Start-Programs' menu in windows select the 'NCHRP Viewer'
- Open the database that you have unzipped and select the processes that you want to review (see Figure G-3). Depending on the size of the database and the speed of your machine, this could take some time.
- A help file is provided in 'Help-Contents'. The same help file is available in PDF format at the following location \Software\NCHRP Data Viewer\Disclaimer.pdf (CD 2 of 2)
- Begin browsing through the examples by selecting different report, bridge, and subdomain IDs.

This viewer is provided "as is" and has not been tested as production software. Its uses are limited but provide an example of the power of producing Process 12-50 formatted data.

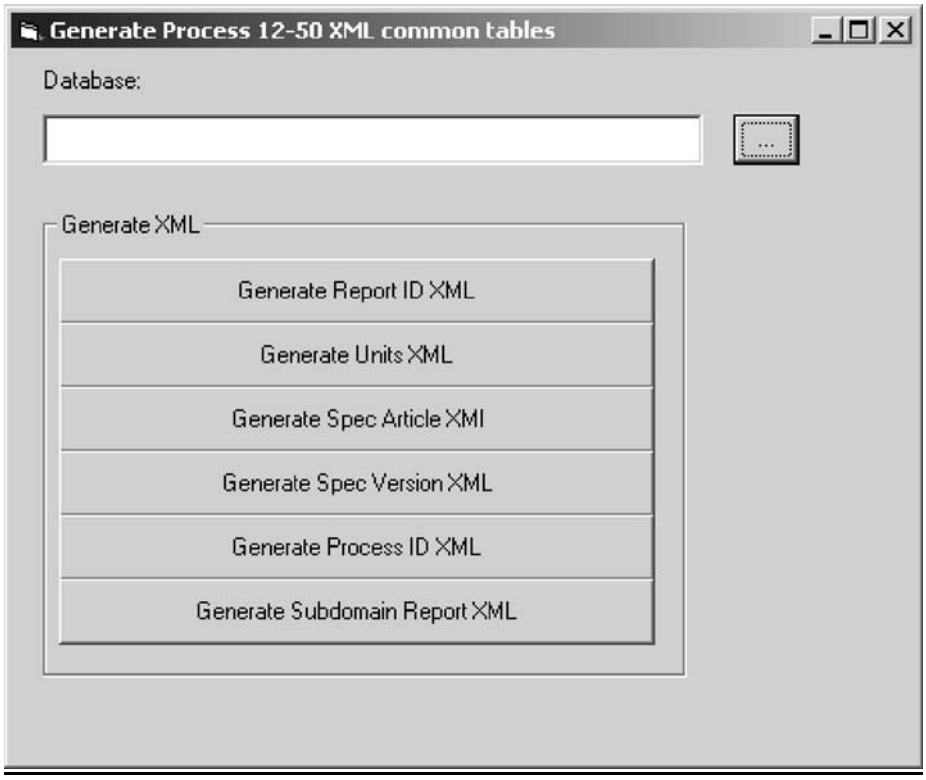

Figure G-2. Common Table XML Generator Utility.

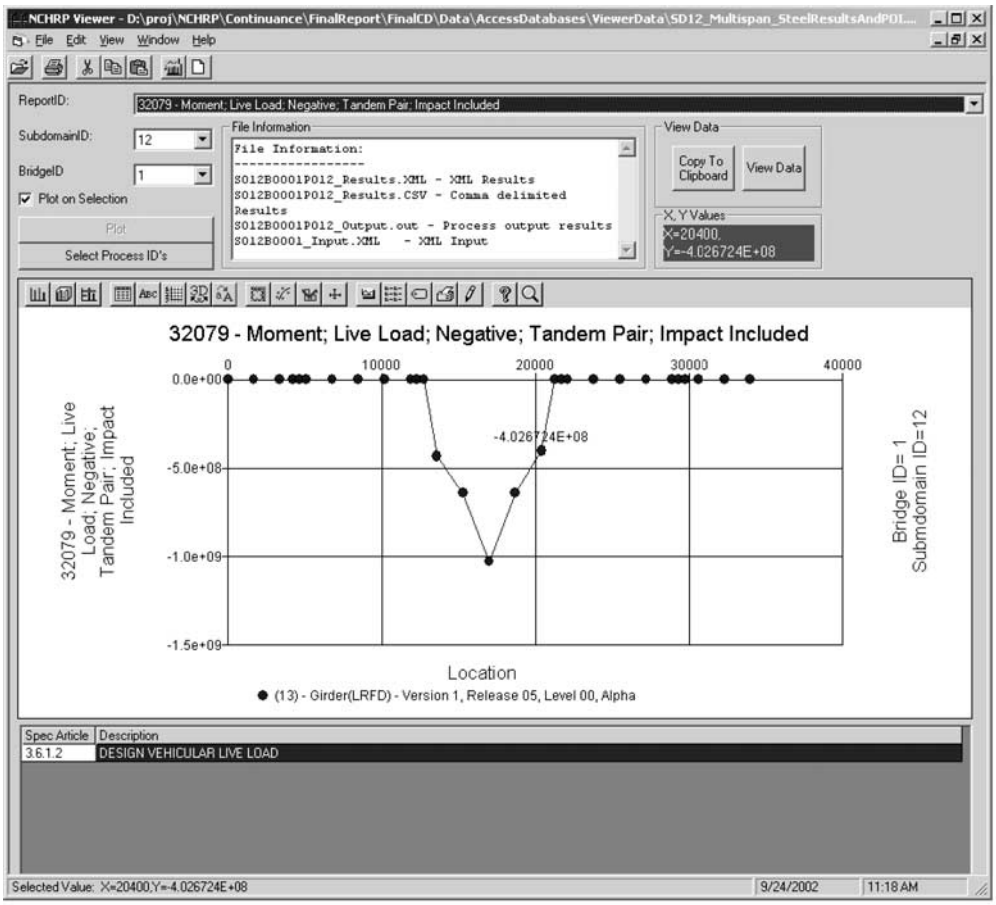

Figure G-3. NCHRP Data Viewer.

‘SubdomainAccessData’ Folder (CD 2 of 2)

This folder contains all of the subdomain data in Microsoft Access format. This data can be viewed using the NCHRP Data Viewer software defined in the previous section.

‘SubdomainRawData’ Folder (CD 1 of 2 and CD 2 of 2)

The Process 12-50 raw data are organized in folders by subdomain number under the main folder heading ‘SubdomainRawData’ (See Figure G-4) A description of the subdomains is provided in Appendix E.

Subdomains 9, 10, 12, 13, and 15 through 20 are also subdivided in subfolders labeled ‘Multispan’ and ‘Simple’. The ‘Simple’ folder contains simple-span bridges reviewed in Phase I of the project while the ‘Multispan’ folder contains bridges of two or more spans created in Phase II of the project.

As new subdomains are defined, additional folders would be added to the ‘SubdomainRawData’ directory.

File Naming Convention

A file naming convention, using the subdomain, bridge and process Ids, is used to organize the data within each subdomain folder. The following file naming convention is used for all of the Process 12-50 data files delivered for this project.

`SsssBbbbb[Pppp]_(type).ext`

Where

`Ssss`	Subdomain ID (e.g. S018)
`Bbbbb`	Bridge ID (e.g. B0202)
`Pppp`	Process ID (Not needed for input unless referring to a specific process) (e.g. P012)
`Type`	Type of file Results – primary Process 12-50 results file POI – points of interest file Input – input file Output – Standard Output from the program
`Ext`	Extension indicating file type XML – xml format CSV – comma-delimited format

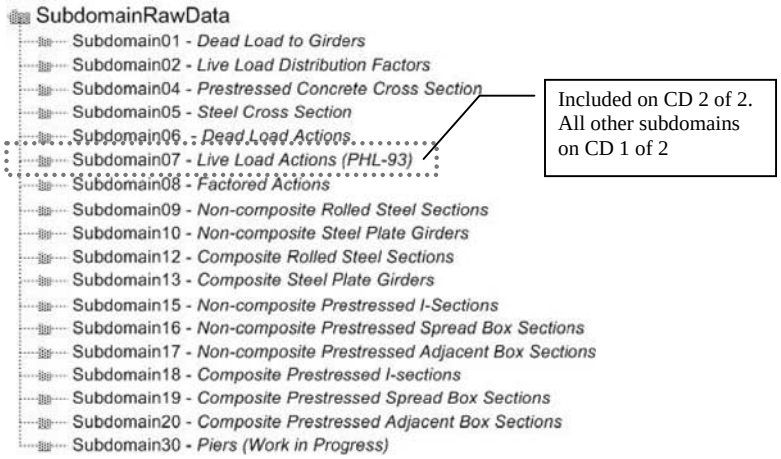

Figure G-4. Subdomain Directory Structure.

INP – input data file for the specific process or subdomain
OUT – output file for the specific process

Examples:

The file '**`S018B0022P008_Results.XML`**' contains the results for Subdomain 18 (Composite Prestressed I-sections), Bridge 22, and Process 8 (BRASS Girder LRFD version 1, Release 5) in an XML format.

The file '**`S018B0022P008_Results.CSV`**' contains the same results in a CSV format.

The file '**`S018B0022_Input.XML`**' contains the input for Subdomain 18 (Composite Prestressed I-sections) and Bridge 22. The process id is not included because this represents the input independent of the process.

The file '**`S018B0022P008_Results.OUT`**' contains the results for Subdomain 18 (Composite Prestressed I-sections), Bridge 22, and Process 8 (BRASS Girder LRFD version 1, Release 5) in a file produced by the process (i.e. the BRASS output file).

Compressed File Conventions

Because of the number and size of the files being delivered, they have been organized in compressed zipped files. The zipped files have been divided by subdomain and file type with names that describe the content. Below is a sample listing for Subdomain 8. Each 'Subdomainxx' directory folder contains files with a similar naming convention. The exception is 'Subdomain30' (pier), which is described in the following section.

SD08_InputINP.zip	Process input files (e.g. files used by the specific process).
SD08_InputXML.zip	Subdomain input files (note these are not process dependent). This file also includes a CSV file that contains input for all of the bridges in the subdomain.
SD08_OutputOUT.zip	Process output files.
SD08_POICSV.zip *	POI files produced by the processes in CSV format.
SD08_POIXML.zip *	POI files produced by the processes in XML format.
SD08_ResultsCSV.zip	Results files produced by the processes in CSV format.
SD08_ResultsXML.zip	Results files produced by the processes in XML format.

*Note: Not all subdomains contain POI data as some processes do not produce this information.

Note on Subdomain 30

Subdomain 30 deals with bridge substructures. Because of its complexity and the lack of two automated processes, this subdomain remains a work in progress. One of the key issues being investigated with Subdomain 30 is the use of a deeper database hierarchy. The results in the subdomain are in a format that uses fewer Report IDs; however, these Report ID auxiliary tables link to other information about the data. For example, the same Report ID (70000) refers to the moment and shear in a footing, but the associated record in the Auxiliary table references a separate Actions table that defines moment, shear, axial, and so forth.

Because only one process was automated for this subdomain, it was difficult to determine the effects of implementing a deeper hierarchy developed for one process into a different process.

Work on this subdomain began by following the same pattern used for the other subdomains. A series of Report IDs

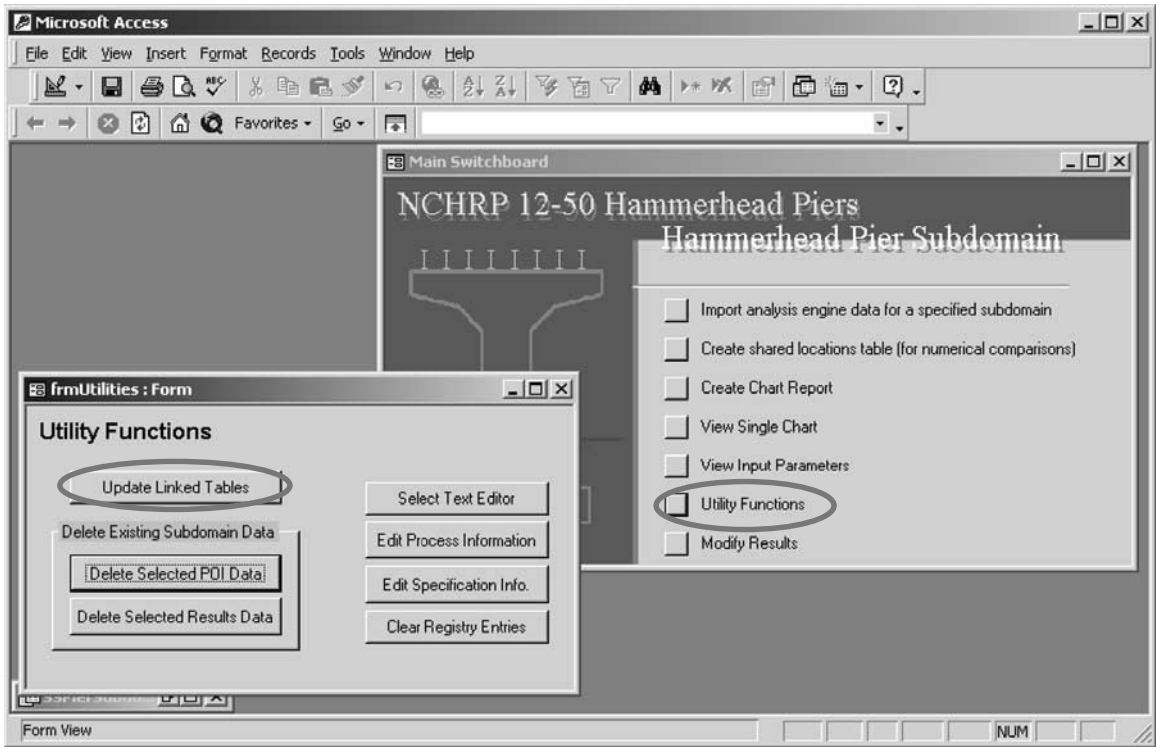

Figure G-5. Subdomain 30 database.

was developed to outline most of the important information about a bridge pier. As work progressed, this series of Report ID's was replaced by the deeper database hierarchy.

As mentioned previously, only one process was automated to produce results in the NCHRP 12-50 format (BRASS-PIER). Therefore, the only piers in the database are the piers for which manually entered results from PAPIER were calibrated. Calibration of BRASS-PIER and PAPIER was just completed at the end of the project, so only a few datasets are available.

The information for Subdomain 30 is provided in the folder 'Subdomain30' and includes the raw data information as well as a zipped file SD30.zip. This file contains all of the raw data in a predefined tree structure.

Subdomain 30 Installation Instructions

For the databases to work properly, the following steps should be followed:

- Install the program NCHRP Viewer from CD 2 of 2 by executing the following \Software\NCHRPDataViewer\Setup.exe.
- This will install the NCHRP Viewer. (**Because of the experimental data hierarchy created for the pier, the Microsoft Access database for the Pier subdomain does not work with the NCHRP viewer provided on this CD. However, installing the viewer will install Graphics Server, which is needed to run the Subdomain 30 database).**
- The SD30.Zip file contained on CD 1 of 2 in folder \SubdomainRawData\Subdomain30 must be installed on the C drive root directory. Once installed, the user can begin reviewing the Subdomain 30 data by loading the file c:\nchrp12-50\SubDomData\Databases\SSPierSubdomain2000.mdb.
- On the 'Main Switchboard' menu click on the 'Utility Functions' button and then on the 'Update Linked Tables' button (see Figure G-5).
- Choose the 'View Single Chart' option from the Main Switchboard to begin browsing the database.

See Appendix E for more detail on the database structure of the Piers subdomain. The types of problems analyzed, modeling assumptions, loadings, and the results of output comparisons between BRASS-PIER and PAPIER are also discussed.

Abbreviations used without definitions in TRB publications:

AASHO	American Association of State Highway Officials
AASHTO	American Association of State Highway and Transportation Officials
ASCE	American Society of Civil Engineers
ASME	American Society of Mechanical Engineers
ASTM	American Society for Testing and Materials
FAA	Federal Aviation Administration
FHWA	Federal Highway Administration
FRA	Federal Railroad Administration
FTA	Federal Transit Administration
IEEE	Institute of Electrical and Electronics Engineers
ITE	Institute of Transportation Engineers
NCHRP	National Cooperative Highway Research Program
NCTRP	National Cooperative Transit Research and Development Program
NHTSA	National Highway Traffic Safety Administration
SAE	Society of Automotive Engineers
TCRP	Transit Cooperative Research Program
TRB	Transportation Research Board
U.S.DOT	United States Department of Transportation