DETAILED DESIGN FOR A MIS FOR THE
SOUTHERN CALIFORNIA RAPID TRANSIT DISTRICT

OCTOBER 1980

PREPARED FOR:
U.S. DEPARTMENT OF TRANSPORTATION
URBAN MASS TRANSPORTATION ADMINISTRATION
Office of Technology Development and Deployment
Washington, D.C. 20590
This document was disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof. The United States Government does not endorse products or manufacturers. Trade or manufacturer's names appear herein solely because they are considered essential to the object of this report.
Detailed Design for a MIS for the Southern California Rapid Transit District

An Automatic Vehicle Monitoring System (AVM) being installed in Los Angeles will record a large amount of operational data that can later be used for management reports. The software required to provide this function is described and structured pseudo-code of the processes and detailed file descriptions are given.
# METRIC CONVERSION FACTORS

## Approximate Conversions to Metric Measures

<table>
<thead>
<tr>
<th>Symbol</th>
<th>When You Know</th>
<th>Multiply by</th>
<th>To Find</th>
<th>Symbol</th>
</tr>
</thead>
</table>

### LENGTH

| m | inches | 2.5 | centimeters | cm |
| ft | feet | 30 | centimeters | cm |
| yd | yards | 0.9 | meters | m |
| mi | miles | 1.6 | kilometers | km |

### AREA

| in² | square inches | 6.5 | square centimeters | cm² |
| ft² | square feet | 0.09 | square meters | m² |
| yd² | square yards | 0.8 | square meters | m² |
| mi² | square miles | 2.6 | square kilometers | km² |
| ac | acres | 0.4 | hectares | ha |

### MASS (weight)

| oz | ounces | 28 | grams | g |
| lb | pounds | 0.45 | kilograms | kg |
| sh tn | short tons | 0.9 | (2000 lbs) | t |

### VOLUME

| tsp | teaspoons | 5 | milliliters | ml |
| Tbsp | tablespoons | 15 | milliliters | ml |
| fl oz | fluid ounces | 30 | milliliters | ml |
| c | cups | 0.24 | liters | l |
| pt | pints | 0.47 | liters | l |
| qt | quarts | 0.95 | liters | l |
| gal | gallons | 3.8 | liters | l |
| yd³ | cubic feet | 0.03 | cubic meters | m³ |
| ft³ | cubic yards | 0.76 | cubic meters | m³ |

### TEMPERATURE (exact)

°F | Fahrenheit temperature | 5/9 latter | Celsius temperature | °C |

°F = °C + 32

°C = (°F - 32) × 5/9
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF ILLUSTRATIONS</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
</tbody>
</table>

## 1. INTRODUCTION  
1-1

## 2. PROGRAM DESCRIPTIONS  
2-1

### 2.1 Form AVM Files  
2-1

#### 2.1.1 Gould Fortran Programs  
2-1

##### 2.1.1.1 Input  
2-10

##### 2.1.1.2 Output  
2-10

##### 2.1.1.3 Processing  
2-11

#### 2.1.2 Select and Sort  
2-11

##### 2.1.2.1 Select  
2-11

##### 2.1.2.1.1 Input  
2-11

##### 2.1.2.1.2 Output  
2-11

##### 2.1.2.1.3 Processing  
2-11

##### 2.1.2.2 Sort  
2-11

##### 2.1.2.2.1 Input  
2-11

##### 2.1.2.2.2 Output  
2-11

##### 2.1.2.2.3 Processing  
2-14

#### 2.1.3 Update Perf Tapes  
2-14

##### 2.1.3.1 Form TP-Trip File  
2-14

##### 2.1.3.1.1 Input  
2-14

##### 2.1.3.1.2 Output  
2-14

##### 2.1.3.1.3 Processing  
2-14

##### 2.1.3.2 Form New Summary Tape  
2-15

##### 2.1.3.2.1 Form Daily Aggregate File  
2-15

##### 2.1.3.2.1.1 Form Daily Section 15 Aggregate File  
2-18
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.3.2.1.1 Input</td>
<td>2-19</td>
</tr>
<tr>
<td>2.1.3.2.1.2 Output</td>
<td>2-19</td>
</tr>
<tr>
<td>2.1.3.2.1.3 Processing</td>
<td>2-19</td>
</tr>
<tr>
<td>2.1.3.2.1.2 Form Daily Run Aggregate File</td>
<td>2-19</td>
</tr>
<tr>
<td>2.1.3.2.1.2.1 Input</td>
<td>2-19</td>
</tr>
<tr>
<td>2.1.3.2.1.2.2 Output</td>
<td>2-19</td>
</tr>
<tr>
<td>2.1.3.2.1.2.3 Processing</td>
<td>2-20</td>
</tr>
<tr>
<td>2.1.3.2.1.3 Form Daily Ridership Aggregate File</td>
<td>2-20</td>
</tr>
<tr>
<td>2.1.3.2.1.3.1 Input</td>
<td>2-20</td>
</tr>
<tr>
<td>2.1.3.2.1.3.2 Output</td>
<td>2-21</td>
</tr>
<tr>
<td>2.1.3.2.1.3.3 Processing</td>
<td>2-21</td>
</tr>
<tr>
<td>2.1.3.2.1.4 Form Daily Schedule Deviation Aggregate File</td>
<td>2-21</td>
</tr>
<tr>
<td>2.1.3.2.1.4.1 Input</td>
<td>2-21</td>
</tr>
<tr>
<td>2.1.3.2.1.4.2 Output</td>
<td>2-24</td>
</tr>
<tr>
<td>2.1.3.2.1.4.3 Processing</td>
<td>2-24</td>
</tr>
<tr>
<td>2.1.3.2.1.5 Form Daily Overload Aggregate File</td>
<td>2-24</td>
</tr>
<tr>
<td>2.1.3.2.1.5.1 Input</td>
<td>2-24</td>
</tr>
<tr>
<td>2.1.3.2.1.5.2 Output</td>
<td>2-24</td>
</tr>
<tr>
<td>2.1.3.2.1.5.3 Processing</td>
<td>2-24</td>
</tr>
<tr>
<td>2.1.3.2.1.6 Form Daily Layover Aggregate File</td>
<td>2-24</td>
</tr>
<tr>
<td>2.1.3.2.1.6.1 Input</td>
<td>2-25</td>
</tr>
<tr>
<td>2.1.3.2.1.6.2 Output</td>
<td>2-25</td>
</tr>
<tr>
<td>2.1.3.2.1.6.3 Processing</td>
<td>2-25</td>
</tr>
<tr>
<td>2.1.3.2.2 Update Summary Files</td>
<td>2-25</td>
</tr>
<tr>
<td>2.1.3.3 Form New Perf Data Base Tape</td>
<td>2-26</td>
</tr>
<tr>
<td>2.1.3.3.1 Input</td>
<td>2-26</td>
</tr>
<tr>
<td>2.1.3.3.2 Output</td>
<td>2-26</td>
</tr>
<tr>
<td>2.1.3.3.3 Processing</td>
<td>2-26</td>
</tr>
<tr>
<td>2.1.3.4 Form Daily Exception Reports</td>
<td>2-26</td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS (CONTINUED)

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.3.4.1</td>
<td>Form Daily Schedule Deviation Exception Report</td>
<td>2-27</td>
</tr>
<tr>
<td>2.1.3.4.1.1</td>
<td>Input</td>
<td>2-27</td>
</tr>
<tr>
<td>2.1.3.4.1.2</td>
<td>Output</td>
<td>2-27</td>
</tr>
<tr>
<td>2.1.3.4.1.3</td>
<td>Processing</td>
<td>2-27</td>
</tr>
<tr>
<td>2.1.3.4.2</td>
<td>Form Daily Overload Exception Report</td>
<td>2-27</td>
</tr>
<tr>
<td>2.1.3.4.2.1</td>
<td>Input</td>
<td>2-27</td>
</tr>
<tr>
<td>2.1.3.4.2.2</td>
<td>Output</td>
<td>2-27</td>
</tr>
<tr>
<td>2.1.3.4.2.3</td>
<td>Processing</td>
<td>2-28</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Update Trouble Tape</td>
<td>2-28</td>
</tr>
<tr>
<td>2.1.4.1</td>
<td>Sort</td>
<td>2-28</td>
</tr>
<tr>
<td>2.1.4.1.1</td>
<td>Input</td>
<td>2-28</td>
</tr>
<tr>
<td>2.1.4.1.2</td>
<td>Output</td>
<td>2-28</td>
</tr>
<tr>
<td>2.1.4.1.3</td>
<td>Processing</td>
<td>2-28</td>
</tr>
<tr>
<td>2.1.4.2</td>
<td>Form Output Trouble Reports</td>
<td>2-28</td>
</tr>
<tr>
<td>2.1.4.2.1</td>
<td>Input</td>
<td>2-28</td>
</tr>
<tr>
<td>2.1.4.2.2</td>
<td>Output</td>
<td>2-29</td>
</tr>
<tr>
<td>2.1.4.2.3</td>
<td>Processing</td>
<td>2-29</td>
</tr>
<tr>
<td>2.1.4.3</td>
<td>Form New Trouble Tape</td>
<td>2-29</td>
</tr>
<tr>
<td>2.1.4.3.1</td>
<td>Input</td>
<td>2-29</td>
</tr>
<tr>
<td>2.1.4.3.2</td>
<td>Output</td>
<td>2-29</td>
</tr>
<tr>
<td>2.1.4.3.3</td>
<td>Processing</td>
<td>2-29</td>
</tr>
<tr>
<td>2.2</td>
<td>Form Summary Reports</td>
<td>2-29</td>
</tr>
<tr>
<td>3</td>
<td>PROGRAM LOGIC</td>
<td>3-1</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>FILE CONTENTS</td>
<td>A-1</td>
</tr>
<tr>
<td>APPENDIX B</td>
<td>DETAILED LOGIC</td>
<td>B-1</td>
</tr>
<tr>
<td>B.1</td>
<td>Form AVM Files</td>
<td>B-2</td>
</tr>
<tr>
<td>B.1.1</td>
<td>Gould Fortran Programs</td>
<td>B-2</td>
</tr>
<tr>
<td>B.1.2</td>
<td>Select and Sort</td>
<td>B-2</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (Concluded)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1.3 Update Perf Tapes</td>
<td>B-2</td>
</tr>
<tr>
<td>B.1.3.1 Form TP-Trip File</td>
<td>B-3</td>
</tr>
<tr>
<td>B.1.3.2 Form New Summary Tape</td>
<td>B-6</td>
</tr>
<tr>
<td>B.1.3.2.1 Information common to Aggregate programs</td>
<td>B-6</td>
</tr>
<tr>
<td>B.1.3.2.1.1 Form Daily-Section-15-Aggregate File</td>
<td>B-8</td>
</tr>
<tr>
<td>B.1.3.2.1.2 Form Daily-Run-Aggregate File</td>
<td>B-11</td>
</tr>
<tr>
<td>B.1.3.2.1.3 Form Daily-Ride-Aggregate Files</td>
<td>B-15</td>
</tr>
<tr>
<td>B.1.3.2.1.4 Form Daily-Shed-Dev-Aggregate File</td>
<td>B-20</td>
</tr>
<tr>
<td>B.1.3.2.1.5 Form Daily-Overload-Aggregate File</td>
<td>B-23</td>
</tr>
<tr>
<td>B.1.3.2.1.6 Form Daily-Layover-Aggregate File</td>
<td>B-26</td>
</tr>
<tr>
<td>B.1.3.2.2 Update</td>
<td>B-29</td>
</tr>
<tr>
<td>B.1.3.3 Form New-Perf-Data-Base Tape</td>
<td>B-31</td>
</tr>
<tr>
<td>B.1.3.4 Form Daily Exception Reports</td>
<td>B-32</td>
</tr>
<tr>
<td>B.1.3.4.1 Form Daily Schedule Deviation Report</td>
<td>B-32</td>
</tr>
<tr>
<td>B.1.3.4.2 Form Daily Overload Report</td>
<td>B-34</td>
</tr>
<tr>
<td>B.2 Common Report Program Contents (reports 2.1 through 2.6)</td>
<td>B-36</td>
</tr>
</tbody>
</table>
LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>MIS Data Flow</td>
<td>2-2</td>
</tr>
<tr>
<td>2-2</td>
<td>Process 1: Form AVM Files</td>
<td>2-3</td>
</tr>
<tr>
<td>2-3</td>
<td>Process 2: Form Output Reports</td>
<td>2-4</td>
</tr>
<tr>
<td>2-4</td>
<td>Process 1.2: Select and Sort</td>
<td>2-5</td>
</tr>
<tr>
<td>2-5</td>
<td>Process 1.3: Update Perf Tapes</td>
<td>2-6</td>
</tr>
<tr>
<td>2-6</td>
<td>Process 1.3.2: Form New Summary Tape</td>
<td>2-7</td>
</tr>
<tr>
<td>2-7</td>
<td>Process 1.4: Update Trouble Tape</td>
<td>2-8</td>
</tr>
<tr>
<td>2-8</td>
<td>Process 1: Expanded</td>
<td>2-9</td>
</tr>
<tr>
<td>2-9</td>
<td>Summary Tape Layout</td>
<td>2-16</td>
</tr>
<tr>
<td>2-10</td>
<td>Basic Standee Computations</td>
<td>2-22</td>
</tr>
<tr>
<td>2-11</td>
<td>Standee Cycle Illustration</td>
<td>2-23</td>
</tr>
<tr>
<td>2-12</td>
<td>Example of Sequence of Report Data</td>
<td>2-30</td>
</tr>
</tbody>
</table>

LIST OF TABLES

<table>
<thead>
<tr>
<th>Table Numbers</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Block Header Record (Tracked Bus File)</td>
<td>2-12</td>
</tr>
<tr>
<td>2-2</td>
<td>Bus Stop Record (Tracked Bus File)</td>
<td>2-13</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

A demonstration of an Automatic Vehicle Monitoring (AVM) system, sponsored by the Urban Mass Transportation Administration (UMTA), is being performed in Los Angeles, with approximately 200 buses and four lines of the Southern California Rapid Transit District (SCRTD). Although the primary purpose of the system is to determine the effectiveness of various on-line control strategies, a large amount of data is recorded daily and can be used to provide management information. A previous Working Paper (WP-79W00738) detailed requirements for MIS reports using AVM data, and a later letter (W24-5026) presented a preliminary design that would provide the required data.

This document expands the preliminary design to a detailed design. In Section 2, the latest data flow diagrams are given, and the inputs, outputs, and processing required by each process are described. Section 3 discusses the structured approach and conventions of the pseudo-code used to describe the processing. Appendix A alphabetically presents the contents of the files and some of the tables referred to in Section 2 and Appendix B presents the detailed pseudo-code described in Section 3.

The contents of this document include revisions based on discussion with SCRTD personnel of an earlier draft copy; it is anticipated that further refinements will result during the implementation phase.
2. PROGRAM DESCRIPTIONS

The programs required to process the AVM log tape and eventually provide MIS reports are described here. The general format is to discuss the required inputs to each program, the resultant outputs and the processing required. The programs follow from the data flow diagrams developed during the preliminary design, although some of the processes in that document have been further decomposed and some modifications have been made. Figure 2-1 shows the first level processes, Figure 2-2 and 2-3 show the second level, and Figures 2-4 through 2-7 show the third and lower levels. Figure 2-8 shows an expansion of Process 1: Form AVM Files.

2.1 Form AVM Files

To summarize the processes involved (see Figure 2-2), the daily AVM log tape is processed by Gould Fortran programs, forming data files which are further selected, sorted and processed by Cobol programs. A research tape is formed including all of the AVM data bus stop; this will probably require a new tape every day. A daily TP-trip file is formed, including, for each time point on each trip, the basic information that is needed to later provide MIS reports. For each report that is to be provided, a separate daily aggregate file is formed from the TP-trip file and is used to update a summary tape, which maintains statistical data from the beginning of the desired time period to the present. At any time, the summary tape can be accessed by report programs to format and provide final totals.

In general, the program names used are the same as the processes on the data flow diagrams. As there are a large number of files involved, and they are referred to not only in this section, but also in the next, the details are not given here, but are provided in the Appendix, where the files and some tables are presented in alphabetical order.

2.1.1 Gould Fortran Programs

A set of programs has been developed by Gould to read the AVM log tape and form a set of files that they will later use to determine the effectiveness of various control strategies. Although the programs were not originally intended for use at SCRTD, a Fortran compiler is available on the Univac system, and it should be possible to use the same programs perhaps with a few modifications to accommodate differences between the different Fortran implementations. Although all the data
FIGURE 2-1. MIS DATA FLOW

2-2
FIGURE 2-2. PROCESS 1: FORM AVM FILES
FIGURE 2-3. PROCESS 2: FORM OUTPUT REPORTS
FIGURE 2-4. PROCESS 1.2: SELECT AND SORT

2-5
FIGURE 2-5. PROCESS 1.3: UPDATE PERF TAPES
FIGURE 2-6. PROCESS 1.3.2: FORM NEW SUMMARY TAPE
Trouble - Reports

Sort By Trouble Report # 1.4.1

Format and Output Reports 1.4.2

Form New Trouble Tape 1.4.3

old-trouble-DB-tape

new-trouble-DB-tape

Daily-Reports

FIGURE 2.7. PROCESS 1.4: UPDATE TROUBLE TAPE
available from these programs are not needed to satisfy reporting requirements, it is less complicated to use the existing programs to form files which will then be read by Cobol programs developed by SCRTD than to build a new set of programs from scratch to provide just the data needed. Also, all the data provided by one of the Gould files will be sorted and copied onto a tape which will be available for later analysis by researchers.

2.1.1.1 Input

The AVM log tape is the primary input. It is possible that some control data will also need to be provided.

2.1.1.2 Output

Four types of files can be provided. Two types of files are appropriate; the other two, which are not yet defined in detail, are not needed here and will not be output. This will be accomplished by skipping calls in the main program or by providing dummy JCL when the job is run. The files useful for MIS purposes, the Trouble Report File and the Tracked Bus Data File, will be stored on disk for later processing. The Trouble Report File consists of all closed trouble reports, written in 1200 (2-byte) word blocks, each block containing a separate trouble report. Since trouble reports may not appear sequentially (by number) on the raw tape, the first block of this file will contain block pointers indexed by trouble report number (i.e., word 13 of block 1 contains the block number of trouble report 13). The Tracked Bus Data File contains a time history of bus transactions at each stop on each AVM line. It is organized into 2,040 word blocks; each block contains 51 logical records 40 words in length. There are four record types:

1. Block Header Record--First 40 words of each block, providing common information,

2. Bus Stop Record--Primary data on a bus transaction at a stop,

3. Bus Assignment/Pullout Record--Data on line/run assignments, pullouts and pullins, and

4. Dispatcher Action Record--Describes completed AVM function key actions by dispatcher.
Each of these records is defined in detail in Table 2-1 and Table 2-2. Records are not sorted and appear in the order in which the event occurred regardless of record type or context.

2.1.1.3 Processing

The details of the processing have been provided by Gould to SCRTD and are not within the scope of this document.

2.1.2 Select and Sort

Although the data in the Tracked Bus File are in time sequential order, later processing is facilitated if the data records are ordered, from major to minor: bus line, bus direction, bus trip and time at bus stop. Rather than do one large sort on what may be over 80,000 records per day, a two-step process will be performed.

2.1.2.1 Select

2.1.2.1.1 Input

The Tracked Bus Data File, previously described, is the input.

2.1.2.1.2 Output

Four files will be created, one for each AVM line, called here Bus Stop By Line. The four files will be copied, one after the other to form the research tape.

2.1.2.1.3 Processing

The first Block Header Records read will provide data for the Bus Stop Record Header. Later ones will be ignored as will Bus Assignment/Pullout Records and Dispatcher Action Records; only Bus Stop Records will be read and written into the appropriate file.

2.1.2.2 Sort

2.1.2.2.1 Input

Each of the four Bus Stop By Line files will be used as input, one at a time.

2.1.2.2.2 Output

A single file, the Ordered Bus Stop File.
### TABLE 2-1 - BLOCK HEADER RECORD (TRACKED BUS FILE)

**BLOCK HEADER RECORD (1ST 40 WDS OF EACH BLOCK)**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>MNEMONIC</th>
<th>UNITS</th>
<th>FORMAT</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time of Day</td>
<td>TOD1</td>
<td>I2</td>
<td></td>
<td>Starting Time (AVM System Time) Of This Block</td>
</tr>
<tr>
<td></td>
<td>TOD2</td>
<td>I2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TOD3</td>
<td>I2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>MO</td>
<td>I2</td>
<td></td>
<td>1-12</td>
</tr>
<tr>
<td></td>
<td>DAY</td>
<td>I2</td>
<td></td>
<td>1-31</td>
</tr>
<tr>
<td></td>
<td>YR</td>
<td>I2</td>
<td></td>
<td>79, 80, 81</td>
</tr>
<tr>
<td>Day of Week</td>
<td>A4</td>
<td></td>
<td></td>
<td>M, T, W, T, F, S, S</td>
</tr>
<tr>
<td>Weather</td>
<td>A2</td>
<td></td>
<td></td>
<td>CL, RA, CY, FG</td>
</tr>
<tr>
<td>AVM Test Period</td>
<td></td>
<td>I2</td>
<td></td>
<td># of 2 wk TP Since Start of Exps.</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Day of TP</td>
<td></td>
<td>I2</td>
<td></td>
<td># of Day this TP (1-14)</td>
</tr>
<tr>
<td>2</td>
<td>Line Number</td>
<td>I3</td>
<td></td>
<td>RTD Line Number</td>
</tr>
<tr>
<td>3</td>
<td>Schedule Number</td>
<td>I5</td>
<td></td>
<td>RTD Schedule Number</td>
</tr>
<tr>
<td>4</td>
<td>AVM Configuration</td>
<td>I2</td>
<td></td>
<td>1-4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SOURCE: Gould, Inc.
<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>MNEMONIC</th>
<th>UNITS</th>
<th>FORMAT</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time of Day</td>
<td>TOD1</td>
<td>Hrs.</td>
<td>12</td>
<td>Military Time</td>
</tr>
<tr>
<td></td>
<td>TOD2</td>
<td>Mins.</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TOD3</td>
<td>Secs.</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Line #</td>
<td>LI</td>
<td></td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Run #</td>
<td>RU</td>
<td></td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Bus #</td>
<td>BU</td>
<td></td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Trip #</td>
<td>TRP</td>
<td></td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Direct Code</td>
<td>DIR</td>
<td></td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Origin Code</td>
<td>OC</td>
<td></td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Destination Code</td>
<td>DC</td>
<td></td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Stop #</td>
<td>STP</td>
<td></td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Time Point Code</td>
<td>TPC</td>
<td></td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Sched. Deviation</td>
<td>SDEV</td>
<td>Secs</td>
<td>15</td>
<td>As Adjusted, from TOD</td>
</tr>
<tr>
<td>Sched. Adjustment</td>
<td>SADJ</td>
<td>Secs</td>
<td>15</td>
<td>(As Specified by Dispatcher)</td>
</tr>
<tr>
<td>Sched. Headway</td>
<td>SHDW</td>
<td>Secs</td>
<td>15</td>
<td>As Adjusted</td>
</tr>
<tr>
<td>Headway Deviation</td>
<td>DHDW</td>
<td>Secs</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Passengers On-Board</td>
<td>POB</td>
<td></td>
<td>13</td>
<td>At Arrival at this Stop</td>
</tr>
<tr>
<td>Passengers Alighting</td>
<td>PB</td>
<td></td>
<td>13</td>
<td>At this Stop</td>
</tr>
<tr>
<td>Total Boardings</td>
<td>TB</td>
<td></td>
<td>13</td>
<td>For Trip</td>
</tr>
<tr>
<td>Total Alightings</td>
<td>TA</td>
<td></td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Passengers On-Board Leader</td>
<td>POBL</td>
<td></td>
<td>13</td>
<td>At Arrival at this Stop</td>
</tr>
<tr>
<td>Load Difference</td>
<td>PDIF</td>
<td></td>
<td>13</td>
<td>POB-POBL</td>
</tr>
<tr>
<td>Running Time Increment</td>
<td>RTS</td>
<td>Secs</td>
<td>15</td>
<td>From Last Stop</td>
</tr>
<tr>
<td>Passenger Waiting Time</td>
<td>PWT</td>
<td>Secs</td>
<td>15</td>
<td>PB x (SHDW-DHDW) x 0.4</td>
</tr>
<tr>
<td>Passenger Trip Lot</td>
<td>PTL</td>
<td>T0 Pt</td>
<td>15</td>
<td>PDB x Dist. from last stop</td>
</tr>
<tr>
<td>Passenger Trip Time</td>
<td>PTT</td>
<td>Secs</td>
<td>15</td>
<td>PDB x RTS</td>
</tr>
<tr>
<td>Passenger Trip Delay</td>
<td>PTD</td>
<td>Secs</td>
<td>15</td>
<td>(PB-FA) x SDEV</td>
</tr>
<tr>
<td>Tactical Sit Code</td>
<td>TSC</td>
<td></td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>SFM Position</td>
<td>SFM</td>
<td></td>
<td>11</td>
<td>L-999</td>
</tr>
<tr>
<td>Tactic In Effect</td>
<td>1</td>
<td></td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Other--Uplink/Downlink Flgs</td>
<td>UPL</td>
<td></td>
<td>B36</td>
<td></td>
</tr>
</tbody>
</table>
2.1.2.2.3 Processing

The Univac sort utility will be used to sort the records by direction and trip. As they have already been separated by line the files can simply be concatenated after sorting. As the records are originally in time sequence, after sorting by direction and trip, they would still be in time sequence within trips, if the Univac sort utility were "stable". As it has been learned that the sort is not stable, the sort will have to be by direction, trip and time.

2.1.3 Update Perf Tapes

This is the most complex process of the four second level processes in Form AVM Files. First a file is formed that includes data only at time points, but includes enough data to provide all required output reports. This file is then appended to previous Detailed Data Base files to form a New Perf Data Base Tape. It is also used to form a New Summary Tape, described later, and to form Daily Exception Reports for schedule deviations and for overloading.

2.1.3.1 Form TP-Trip File

2.1.3.1.1 Input

The Ordered Bus Stop File.

2.1.3.1.2 Output

The TP-Trip File, including for each trip, the direction, run and bus numbers, for each time point during the trip, the time at the time point, the adjusted schedule deviation, number of passengers aboard at that time point, number of passengers on and off between the time point and the previous one, the maximum number of passengers aboard at any bus stop between time points and the number of the bus stop where that maximum occurs.

2.1.3.1.3 Processing

The Ordered Bus Stop records are read one at a time and at each bus stop the maximum number of passengers aboard since the last time point is updated, as well as the identification of the maximum passenger bus stop. At each time point, the number of passengers boarding and alighting since the last time point is computed by subtracting, from the latest totals, the totals recorded at the previous time point. The adjusted schedule deviation is taken as the schedule deviation plus the schedule
adjustment. Passengers aboard is taken directly from the bus stop record time. To allow later determination of layover time and run time to the first time point, data from the first and last bus stop records received during a bus trip are also saved, with unique identifiers stored instead of the time points.

2.1.3.2 Form New Summary Tape

This is the most complex process among the second level process in Figure 2-5, and is further decomposed in Figure 2-6.

Basically, a two-part procedure is followed to create each new summary file: first, the daily TP-Trip File is read and the data processed to provide statistics aggregated in the same format as is presently provided on the Old Summary Tape. The Daily Aggregate File is then read along with the Old Summary File of the same type; corresponding data elements (i.e., the same line, direction, day-of-week aggregation, time-of-day aggregation, and time point) are combined, and a New Summary File is written.

Figure 2-9 shows the general layout of the various files and headers on the Summary Tape. It is anticipated that a given summary tape will accumulate data over a monthly period, although other periods could also be used. The Summary Tape will remain essentially the same length after each update, as new data is combined with previous data in each file, not appended to it; therefore there is no real limit on how long a given summary tape can be used to accumulate data. The first day of a summary, the files to be included are selected and the aggregation periods and limits required by each are provided as inputs. That data is then included in the header of each summary file, and is read and used to control the aggregation and selection for the rest of the summary time period. That is, the files selected to be summarized and the criteria to be used for each remain the same for a given summary period, but may be changed at the beginning of the next period.

2.1.3.2.1 Form Daily Aggregate File

A different program of this type is required for each of: Section 15, Running Time, Ridership (two types), Schedule Deviation, Overloading (in addition to daily Schedule Deviation and Overloading Reports, the data for which are not summarized here).
Summary Tape:

<table>
<thead>
<tr>
<th>Tape Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>File 1 Header</td>
</tr>
<tr>
<td>File 1</td>
</tr>
<tr>
<td>File 2 Header</td>
</tr>
<tr>
<td>File 2</td>
</tr>
<tr>
<td>File n Header</td>
</tr>
<tr>
<td>File n</td>
</tr>
</tbody>
</table>

FIGURE 2-9. SUMMARY TAPE LAYOUT
In general, all programs of this type operate in the same manner and provide an output similar to that shown in the Appendix for the summary files. The weather-condition and school-condition pertain to the entire day's operation. The day-interval and time-interval refer to the manner in which the data are aggregated and will be further described below.

To allow flexibility in choice of aggregation period, an approach using user-defined table input has been chosen. The layout of the time-of-day (TOD) table is shown in the Appendix. A TOD code first indicates if aggregation by time period is desired, or whether all trips are to be individually saved. If aggregation is to be performed, the table-length tells how many table entries follow. Each entry consists of a finish-time for an interval (the start-time is assumed to be the previous finish-time; the first start-time is assumed as 4:00 as SCRTD defines their schedule day to begin at 4:00 A.M. The day and date saved in various headers refer to the beginning of the schedule day). For example, the entries 5:00, 6:00, 7:00, 7:20, 7:40, 8:00 would define:

<table>
<thead>
<tr>
<th>Time-Interval</th>
<th>Times Included</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4:00 5:00</td>
</tr>
<tr>
<td>2</td>
<td>5:00 6:00</td>
</tr>
<tr>
<td>3</td>
<td>6:00 7:00</td>
</tr>
<tr>
<td>4</td>
<td>7:00 7:20</td>
</tr>
<tr>
<td>5</td>
<td>7:20 7:40</td>
</tr>
<tr>
<td>6</td>
<td>7:40 8:00</td>
</tr>
</tbody>
</table>

The time at which a trip begins determines the time-interval into which that trip falls, and summations, averages or maximums are carried out over all data occurring in that time interval. As the TOD-table will be included in the Old Summary File header after its information, the same time intervals will be used to create each later Aggregation File, until the next period begins.

A similar technique is used for day-of-week (DOW) aggregation, only here the days need not be contiguous. The Appendix shows the format of the DOW-table, which has a DOW-code to indicate if there will be aggregations or whether each day should be handled individually; and a table-length if there is to be aggregation.
In order to accommodate all the possibilities of aggregation that may be desired, each table entry includes seven positions, corresponding to day of the week (Monday = 1, Sunday = 7). A non-zero table entry indicates that that day will be included in the aggregation for that day-interval, e.g.,

```
... ...
1 1
1 1 1
1 1
```

would accumulate Monday and Friday as day-interval 1, Wednesday through Thursday as day-interval 2 and Saturday and Sunday as day-interval 3. As each Aggregation File involves only one day's data, there is no DOW aggregation performed during this process; it occurs during the Update Summary process.

The use of table input to define aggregation intervals allows as much flexibility as the user desires, without having to modify the programs.

Although the computations required to form the various daily aggregate files are different, the technique is basically the same in each case. Individual records are read from the TP-Trip File, the necessary computations are performed, sometimes involving data from the previous time point as well as the current one, and the results are used to update the element of a vector corresponding to that time point. When the next record read is for a new trip, the start time is checked to determine if a new time interval has been entered. If not, processing proceeds as before with the same elements being updated by new data; if so, statistics are computed for each vector element written out, and computations for the new time interval are begun. In addition to between-time-point computations, trip totals are computed and stored as the last TP element. In the detailed descriptions that follow, only the computational differences will be discussed.

2.1.3.2.1.1. Form Daily Section 15 Aggregate File

Data on passenger ridership is required yearly in UMTA Section 15 reports. All the needed data can be derived from AVM data, and simulation using actual passenger loadings from SCRTD Line 44 has shown that accuracy requirements can be met using passenger loadings only at time points.
2.1.3.2.1.1 Input

The TP-Trip File.

2.1.3.2.1.2 Output

The S15-Aggregate Files containing passengers boarded, bus miles, passenger miles, bus minutes, passenger minutes, capacity miles, seat miles and bus trips.

2.1.3.2.1.3 Processing

The DOW-code and TOD-code are determined by UMTA reporting requirements: for weekdays, all trips in the AM-peak, midday, PM-peak and night (as specified by SCRTD) are aggregated; for Saturday and Sunday (separately), only all-day totals are used. Individual TP-trip records are read and desired measures are computed and summed until the line, direction or time interval changes, when an output record is written. Passengers boarded is directly available from the input, the average number of passengers aboard between time points is computed from the passengers aboard at this and the previous time points and multiplied by the time and distance between time points to get passenger minutes, passenger miles and bus miles, and a table is referenced to find capacity miles and seat miles. The number of bus trips is incremented each time the first time point of a new trip is read.

2.1.3.2.1.2 Form Daily Run Aggregate File

2.1.3.2.1.2.1 Input

The TP-Trip File.

2.1.3.2.1.2.2 Output

The Daily Run Aggregate File and the Daily Layover Aggregate File. The Run File provides scheduled running time and deviation from scheduled running time between time points, from the start of the trip to the first time point (if the first bus stop on a trip is not a time point), and from the first to last bus stop of a trip. If time-of-day aggregation is specified, mean values are computed for scheduled run time and run time deviation and the standard deviation of the run time is also found. The inclusion of the run time standard deviation allows schedulers to better determine a run time that can be attained by most buses and to determine route segments that may require special service under certain conditions. The Layover File includes time information at the first and last bus stop of each trip.
2.1.3.2.1.2.3 Processing

The basic processing has already been described. The actual run time is computed as the time at this point minus the time at the previous point. The scheduled time at each point is determined by adding the time at each point to the adjusted schedule deviation at that point. The scheduled run time is then computed as the scheduled time at this point minus the scheduled time at the previous point and the run time deviation is computed as the scheduled run time minus the actual run time.

If time-of-day aggregation is being performed, the trip vector accumulates the sums of scheduled run times, run time deviations and run times squared as well as the sample size for each time point. When the time interval, direction or line changes, the averages are computed by dividing the sums by the sample sizes, and the standard deviation of the run time is also computed.

A separate file is used for data that can be used to compute layover information, even though this could be considered to be part of running time data, because layover data could not easily be computed at the same time as the other run time data. That is, layover time is not directly available from the data given; it can be computed by first reordering the TR-Trip Records by line, direction and run number (they are presently ordered by line, direction and trip number). Then, for all trips in each run, the last bus stop record containing actual time and adjusted schedule deviation, will be immediately followed by the first bus stop record of the next trip for that bus and the actual and scheduled layover times can be computed from them. However, rather than resort the whole file, skip most of the records and go through many of the same computations as have already been done to form the Run Aggregate File, it was decided to write out the pertinent records as they are being processed for the Run File. A later sort will be much quicker with fewer and shorter records and desired aggregations can be performed independently of those chosen for the Run File.

2.1.3.2.1.3 Form Daily Ridership Aggregate File

This file is later accessed by two report generating programs to construct detailed and summary reports providing ridership data.

2.1.3.2.1.3.1 Input

The TP-Trip File. A table correlating number of seats and bus capacity with bus number and a table giving distance between time points is also needed.
2.1.3.2.1.3.2 Output

The Daily Ride Aggregate File. It summarizes, at each time point, the number of passengers aboard, the number of passengers on and off, the maximum aboard between time points and the bus stop at which the maximum occurred. It also includes the time with standees, in increments of five persons, and trip totals of passenger trips, passengermiles, passenger-miles per passenger trip, load factor, standee minutes and standee minutes per standee.

2.1.3.2.1.3.3 Processing

The first group of data is directly available from the TP-Trip File. The passenger miles are computed in the same manner as for the Section 15 data, and the same principle is used to compute the standee minutes; that is the number of passengers between time points is assumed to be fit by a straight line between the number of passengers at the time points. The time with standees is determined by the amount of time that the passenger straight line is above the horizontal line representing the number of seats. The area formed by that part of the passenger straight line that is above the horizontal seat line gives standee minutes, and this number divided by the time between time points gives the average number of standees.

Figure 2-10 shows the relationship of the basic quantities involved, for conditions in which there are standees.

The computation of standee minutes per standee is more complex, as it first requires the determination of the average number of standees during a "standee cycle," which begins whenever there are standees and ends whenever there are no standees remaining. In Figure 2-11, there are three standee cycles. At the end of each cycle the average number of standees is found by dividing the standee minutes in the cycle by the total time with standees.

2.1.3.2.1.4 Form Daily Schedule Deviation Aggregate File

This file stores the sum of the number of schedule deviations in one-minute intervals for all trips in the chosen time intervals. When the output report is later run, the numbers are converted to percentages.

2.1.3.2.1.4.1 Input

The TP-Trip file.
FIGURE 2-10. BASIC STANDEE COMPUTATIONS
FIGURE 2-11. STANDEE CYCLE ILLUSTRATION
2.1.3.2.1.4.2 Output

The Schedule Deviation Aggregate File including, for each time point, a 22 element vector storing schedule deviation histogram data from minus to plus ten minutes, (including a less than -10 element and a greater than +10 element). The limits can be changed if desired.

2.1.3.2.1.4.3 Processing

Individual TP-trip records are read and the adjusted schedule deviation is examined to determine which histogram element should be incremented. When line, direction or time-interval changes, an output record is written.

2.1.3.2.1.5 Form Daily Overload Aggregate File

This file, originally specified in WP79W00738, has been changed to provide additional data. It now includes, in a similar fashion to the schedule deviation report just described, load factor data in .1 increments.

2.1.3.2.1.5.1 Input

The TP-Trip File. A table to provide the capacity of each bus is also required.

2.1.3.2.1.5.2 Output

The Daily Overload Aggregate File, including, for each time point, a 21 element vector storing the number of bus trips in the appropriate load-factor interval. The load factor limits can be changed if desired.

2.1.3.2.1.5.3 Processing

Individual TP-trip records are read and the average load between time points is found from the number of passengers aboard at this and the previous time points. The load factor is computed using the bus capacity from the load-table and the appropriate element of the overload histogram is incremented. When line, direction or time-interval changes, an output record is written.

2.1.3.2.1.6 Form Daily Layover Aggregate File

The Layover File formed during the running of the Run program is sorted by line, direction, and run and then used to determine the scheduled layover time and the layover deviation.
2.1.3.2.1.6.1 Input

The Layover File, consisting of only "first-bus-stop" and "last-bus-stop" records from the TP-trip file.

2.1.3.2.1.6.2 Output

The daily Layover Aggregate File.

2.1.3.2.1.6.3 Processing

The records are read one at a time; as the file only includes "first bus stop" and "last bus stop" records for each trip, each time a new record is different from the previous record (unless the run also changes) the scheduled layover time and layover time deviation are computed from the actual times and the adjusted schedule deviations or time interval at each bus stop. The scheduled layover time and layover deviation are added to previous values until the line, direction or time interval changes, when averages are formed and written out.

2.1.3.2.2 Update Summary Files

The format of the data in the daily output aggregate files is the same as that of the old summary tape data. Consequently, a standard "master file update" can be performed. A record is read from each file; if the old summary record key is less than the aggregate record key, the old summary record is written as a new summary record and another old summary record is read. If the keys are the same, the data from the aggregate record is combined with the old summary record, e.g., updating an average or total, and a new aggregate record is read. If the old summary record is greater than the aggregate record, the aggregate record is written as the new summary record and a new aggregate record is read. The key is composed of the line, direction, day-interval, and time-interval of trip (depending on whether time of day aggregation is being performed). As the end of either file is reached, the key for that file is set to "high-values", the highest collating value for the given Cobol implementation and, from the above description, it can be seen that the rest of the remaining file will be read and written. When both keys are high-values, the job is completed. As each aggregate file contains data from a single day, only those summary records corresponding to that day-interval are changed.
The method in which the data is combined depends upon the manner in which the data is presently aggregated. Most of the data are simply sums, e.g., bus miles, sample size, standee time; therefore the combining process simply consists of adding the daily aggregate to the old summary to form the new summary. There is also one case where only a extreme value, the maximum number of passengers aboard between time points, is saved. In this case, a simple comparison between the aggregate maximum and the old summary maximum is used to determine the new summary maximum. (The associated maximum passenger bus stop is also selected at this time.) Other types of aggregation are required to handle averages, used in the cases of run time, layover time and load factor; and variance, used in the case of run time. Slightly more processing is required here to form the new summary average from the old summary average, the aggregate average, and the sample sizes—the algorithm to accomplish this is given in Section 3.

Figure 2-6 shows the contents of the updated files and the method used to combine the aggregate and summary files. The same update program, using different data definitions, should be able to be used to update all of the files.

2.1.3.3 Form New Perf Data Base Tape

2.1.3.3.1 Input

The Old Perf Data Base Tape and the newly formed TP-Trip File are inputs. The Old Perf Data Base Tape includes a tape header telling the dates included, followed by each included day's TP-Trip File and associated date header.

2.1.3.3.2 Output

The New Perf Data Base Tape. It is the same format as the Old Perf Data Base Tape.

2.1.3.3.3 Processing

The old tape header is read, updated and written to the new tape. All of the following records are read and rewritten to the new tape, a date header is formed and written, and the entire TP-Trip File is written after it.

2.1.3.4 Form Daily Exception Reports

These are of two types, schedule deviation and overloading. Each TP-Trip record is checked to determine if the adjusted schedule deviation indicates that a bus is too far ahead of or behind schedule (separate limits for each), or has a passenger
load that is greater than a user-selected load factor. For each out-of-tolerance point, data for the preceding and following buses is also output.

2.1.3.4.1 Form Daily Schedule Deviation Exception Report

2.1.3.4.1.1 Input
The TP-Trip File and user provided early limit and late limit.

2.1.3.4.1.2 Output
The Daily Schedule Deviation Exception Report is printed. The contents of this report include scheduled time and schedule deviation at the time point for the out-of-tolerance trip and for preceding and following trips. An example of an output displaying information of this type, developed by Gould, is shown in the Appendix.

2.1.3.4.1.3 Processing
The TP-Trip File is read one record at a time until three trips have been read. Time point data for each trip are stored in a separate vector. The adjusted schedule deviation at each time point in the second vector is checked against the limits; if a limit is exceeded, the data stored for that time point in each of the three trip vectors are printed. When all of the time points in the second vector have been checked, the contents of the second vector are placed in the first vector, those of the third vector are placed in the second vector, and a new trip is read into the third vector. Modifications in this process are required to account for the first and last trips on a given line and direction.

2.1.3.4.2 Form Daily Overload Exception Report

2.1.3.4.2.1 Input
The TP-Trip File and user-provided load-factor limit. A bus number to bus capacity table is also required.

2.1.3.4.2.2 Output
The Daily Overload Exception Report is printed. The contents of this report include the time-point time, load factor and number of passengers over the load factor limit for the out-of-tolerance trip and for preceding and following trips. Format of this report is similar to the Daily Schedule Deviation Report shown in the Appendix.
2.1.3.4.2.3 Processing

The same general technique used in Schedule Deviation processing, checking the second of three trip vectors for out-of-tolerance condition, is used here. Slightly more processing is required here, however, as different buses may have different capacities and the load factor is specified as a percent of capacity. Therefore, as each trip is read, the capacity of the given bus is looked up in a table. As each time point record is read the maximum passengers between time points is divided by the capacity to determine the maximum load factor.

The load factor is later compared to the load factor limit and information on that time point as well as the preceding and following buses at that time point are printed out.

2.1.4 Update Trouble Tape

The Trouble Report File generated by the Gould Fortran programs, ordered by time, is first sorted by Trouble Report Number. The resultant Ordered Trouble Report File is appended to the Old Trouble Data Base Tape in a similar manner as was done with the Perf Data Base Tape. Based on routing data included in the trouble reports the Ordered Trouble Report File is also used to produce a daily set of output reports which are sent to specified users.

2.1.4.1 Sort

2.1.4.1.1 Input

The Trouble Report File, produced daily by the Gould Fortran programs.

2.1.4.1.2 Output

The Ordered Trouble Report File.

2.1.4.1.3 Processing

Sort by Trouble Report Number.

2.1.4.2 Form Output Trouble Reports

2.1.4.2.1 Input

The Ordered Trouble Report File.
2.1.4.2.2 Output

Daily Trouble Report Reports. The Appendix shows the format of the trouble reports presented on a Dispatcher's display. The same format will be used for the output report.

2.1.4.2.3 Processing

The Trouble Reports will be read and, based on the distribution codes included, written to separate files. Each file will then be read, the data elements formatted as previously shown, and a separate report (which may contain multiple copies) printed for each distribution code.

2.1.4.3 Form New Trouble Tape

2.1.4.3.1 Input


2.1.4.3.2 Output


2.1.4.3.3 Processing

The Old Trouble Tape Header is read, and the dates included are updated with today's date. The header is then written to the New Trouble Report Tape, followed by all the tape records on the Old Trouble Report Tape. A header is created and written with data corresponding to today's parameters, followed by all records in the Ordered Trouble Report File.

2.2 Form Summary Reports

At any time, summary reports may be run from data on the Perf Summary Tape, as the daily update programs incorporate the latest day's date into the previous summary. Depending on the initial choice of day-of-week and time-of-day aggregation each file's data may include anything from each trip for each day of the week (an unlikely choice) to daily aggregations aggregated over the whole week—in effect, one line of data. In general, something in between will be specified, and Figure 2-12 shows the general format and order of the output for: DOW—weekdays, and weekends; TOD—AM peak, midday, PM-peak and night. The
FIGURE 2-12. EXAMPLE OF SEQUENCE OF REPORT DATA
first"page" includes, for line 41, northbound, weekdays, data for each time point by time interval, and a daily total, or average, as the case may be. The next page provides the same data for the same line and direction, but for weekends. The next two pages provide the same order of information for the southbound trips, followed by the other lines, in order. All this data can be provided merely by formatting the data in the summary files. To get a weekly aggregation for each line, direction time interval, etc. in the same order as shown, the data is sorted again, by line, direction, time-interval or trip, and day-interval. The records are now read and aggregated by time point until the line, direction, trip or time-interval changes, at which time the weekly aggregate is written. A simpler technique could be used, storing data in arrays as they are read and aggregating them properly at the end of the file; however, to accommodate the possibility that 2,000 trips may have to be handled requires inordinately large arrays. In any case, as the files will be substantially condensed by the time this processing needs to be done, the time to sort and process should be minimal.

The Schedule 15 and Ride Trip Reports contain data that can be meaningfully combined further, to form line and system totals. That is, passenger miles or standee minutes during the AM-peak for line 41 and for the entire (AVM-equipped) system are valid measures, whereas average run times or schedule deviations have meaning when referred to a particular route configuration. Therefore, these reports will be further aggregated whereas the others will stop as previously described.
3. PROGRAM LOGIC

The logical procedures required to implement the programs described in Section 2 are detailed in Appendix B. In general, a structured approach is used; that is only sequential statements (including Calls), If . . . Else, For i = n1 to n2, Do . . . While or Repeat . . . Until (in the former, the specified condition is tested before the included code is performed; in the latter, the test is performed afterwards) are included. A Cobol-oriented approach is used, although the statements used may not be directly implementable in standard Cobol. One such statement is Assignment By-Name, borrowed from PL/I, to indicate that all of the values of the variables in one data structure that have the same name as variables in the second data structure are replaced by the values of the second set of variables. Too, arrays (tables, in Cobol) are often used to aggregate data for all time points in successive trips until a control variable changes, with separate rows of the array being maintained for time interval aggregation, direction aggregation and line aggregation. As an update to a time interval aggregation is always accompanied by an update to each of the others, this is indicated by replacing the "row" index by *. For example:

\[
\text{calc-run-time}(\ast, \text{new-TP}) = \text{calc-run-time}(\ast, \text{new-TP}) + \text{run-time}
\]

adds "run-time" to "column" new-TP of all "rows" of calc-run-time. When delineating lines of text, * means the included text is comment. The symbols +, x and / refer to addition, multiplication and division; subtraction is spelled out as "minus" due to the possibility of confusion with the many hyphenated variable names. Periods are used to indicate the logical end of compound statements; in more complex instances, "Ends" are also used.

In general, the meaning of the various conventions should be clear when the program descriptions are being read, at least after a few of them have been studied. Also, as much as possible, the logic of the different programs is the same, with only the computations of the pertinent measures being different. This is particularly true with all of the programs of a given type, that is those that form daily-aggregate files are very similar, while the various update and report program descriptions are virtually identical. Even among the different types of programs, a similar program sequence is used, and the determination of control breaks is based on the comparison of "keys" of the new and previous records. A key, defined at the beginning of each program description, is composed of various data that can be used to indicate that the end of a line, direction,
trip, time interval, etc. has been reached, requiring some sort of summary and reinitialization processing. When the end of an input file has been reached, high-values are placed in the key, and are used to control the finish of the program.
APPENDIX A - FILE CONTENTS

The contents of the files described in Section 2 and 3 are defined here, in alphabetical order. In general, each file includes a header, a key and data. Some notational conventions are:

- means is equivalent to
+ means and
[] means either-or
{} means iterations of the component enclosed
() means the enclosed component is optional.

The iteration limits are included below and above the braces, unless they are obvious.

As the format of the data in the Aggregate Files is the same as the (Old and New) Summary Files, only the summary files are given. (The headers are slightly different, as the summary headers also provide data on aggregation limits. Also, instead of writing out the contents of the Report Print Files, examples of the output format are given.)
BASIC-OPERATING-SCHEDULE (AVM Log Tape)

Heading = line + schedule + direction + date-created + date effective +
            time-pt = numbers +
            time-pt-names +
            notes

+ Schedule = \{Trip + Run + Pull-Out-Time +
                Pull-Out-Division + Misc. +
                Schedule-Times + Noclue +
                Next-Code + Next-Time\}
            + ( {subheading/note/blank} )
            + End-of-Direction
            + End-of-Division

NOTE: Ordered by division line, direction,
      heading, schedule, end-of-division
BUS-STOP-BY-LINE

Header + { lines bus-stop data }

Header = file-ID + date + day + weather + school

data = bus-stop-record from tracked-bus-file
DAY OF WEEK (DOW) TABLE

Table-length 7
DOW-code + table-length + \{ indicator (j) \} \\
1 j = 1

Where:

j corresponds to day-of-week:

1 = Monday, 7 = Sunday

indicator (j) = 0 or 1;

data for all days whose indicator = 1
will be aggregated

DOW-code = \begin{cases} \text{ALL, if want all days individually} \\
\text{INT, if want day-of-week} \end{cases}
LAYOVER SUMMARY

Header + { key + data }
   trips

header = file-ID + DOW-table + TOD-table + weather code + school-code

key = line + direction + day-int + (time-int) + (trip)

data = sched-layover + layover-dev + sample-size

NOTE: Either time-int or trip-int present, but not both
ORDERED BUS STOP

Header + 

{ record } }

trips TPs

Header = file-ID + date + day + weather + school

Record = same as tracked-bus record

NOTE: Ordered by line, direction, trip.
OVERLOAD SUMMARY

Header + { key + data }
trips

Header = file-ID + load-factor-threshold + DOW-table
+ TOD-table + weather-code + school-code

key = line + direction + day-int + time-int

2.0

data = TP + { buses-in-load-factor-int }

0

NOTE: .1 load factor increments--20 intervals
+ 1 interval for >2.0.
PERF DATA BASE

end-date
File-Header + { date-header + { record }}
start-date trips TPs

file-header = file-ID + start-date + end-date

date-header = date + day + weather + school

record = same as TP-trip record
RESEARCH TAPE

Identical to Ordered Bus Stop File
**Passenger Loading Report**

**Queen City Metro**

**Route 43**

**Reading Road Inbound**

**Weekday Averages for December 1978**

**Trip Summary**

<table>
<thead>
<tr>
<th>Segment</th>
<th>Lockland to Roselyn</th>
<th>Roselyn to Calhoun</th>
<th>Calhoun to Clinton</th>
<th>Clinton to Lincoln Grove</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule</td>
<td>Run Time</td>
<td>Standee</td>
<td>Run Time</td>
<td>Standee</td>
</tr>
<tr>
<td>Time (min)</td>
<td>1-5 6-10</td>
<td>11-15</td>
<td>1-5 6-10</td>
<td>11-15</td>
</tr>
</tbody>
</table>

**Ride Detailed Report**

*Only those trips with standees are printed*

Source: CM
RIDE DETAILED SUMMARY

leader + { key + data } }

trips TP

header = file-ID + DOW-table + TOD-table +
weather-code + school-code

key = line + direction + day-int + (time-int)
  + (trip)

data = TP + psgrs-boarding-bet-TPS + psgrs-
  alighting-bet-TPs + max pasgrs-bet-TPs +
  \{standee-time\} + sample-size

0-5 standees
5-10 standees
>10 standees
**ROUTE PRODUCTIVITY REPORT**

**QUEEN CITY METRO**

**ROUTE 43**

**HEADING ROAD**

**JLBUSBNG**

**WEEKDAY AVERAGES FOR DECEMBER 1979**

**TIME PERIOD TOTALS**

<table>
<thead>
<tr>
<th>TIME PERIOD</th>
<th>TOTAL ON</th>
<th>TOTAL OFF</th>
<th>NO. OF VEH. TRIPS</th>
<th>SEAT-MILES</th>
<th>SEAT-HOURS</th>
<th>PASS.-MILES</th>
<th>PASS.-HOURS</th>
<th>LOAD FACTOR</th>
<th>OCCUP FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM PEAK 6-9</td>
<td>453</td>
<td>375</td>
<td>6</td>
<td>4813.5</td>
<td>309.6</td>
<td>1929.0</td>
<td>115.4</td>
<td>48.3</td>
<td>47.6</td>
</tr>
<tr>
<td>MIDAY 9-1-14</td>
<td>847</td>
<td>682</td>
<td>14</td>
<td>6967.0</td>
<td>341.7</td>
<td>2953.9</td>
<td>224.8</td>
<td>42.4</td>
<td>39.4</td>
</tr>
<tr>
<td>PM PEAK 15-1-16</td>
<td>475</td>
<td>354</td>
<td>7</td>
<td>3416.7</td>
<td>287.4</td>
<td>1362.0</td>
<td>110.9</td>
<td>30.1</td>
<td>36.4</td>
</tr>
<tr>
<td>NIGHT 16-1-24</td>
<td>236</td>
<td>186</td>
<td>7</td>
<td>3152.7</td>
<td>219.4</td>
<td>864.8</td>
<td>53.9</td>
<td>24.9</td>
<td>24.6</td>
</tr>
<tr>
<td>ALL DAY 6-1-24</td>
<td>2000</td>
<td>1597</td>
<td>36</td>
<td>17922.7</td>
<td>1356.0</td>
<td>7116.5</td>
<td>535.8</td>
<td>39.7</td>
<td>39.4</td>
</tr>
</tbody>
</table>

*LOAD FACTOR = PASS.-MILES/SEAT-MILES X 100%
**OCCUP FACTOR = PASS.-HOURS/SEAT-HOURS X 100%

SOURCE: GM
RIDE TRIP SUMMARY

header + { key + data }
trips

header = file-ID + DOW-table + TOD-table + weather-code + school-code

key = line + direction + day-int + (time-int) + (trip)

data = psgr-trips + psgr-mis + load-factor + standee-mis + sample-size
### QUEEN CITY METRO
**ROUTE 43**
**READING ROAD**
**INBOUND**

**WEEKDAY AVERAGES FOR DECEMBER 1978**

#### TIME PERIOD SUMMARY

<table>
<thead>
<tr>
<th>SEGMENT 1</th>
<th>SEGMENT 2</th>
<th>SEGMENT 3</th>
<th>SEGMENT 4</th>
<th>SEGMENT 5</th>
<th>SEGMENT 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCALAND ROSELMN</td>
<td>ROSELMN-CALIFORN</td>
<td>CALIF - CLINTON SP</td>
<td>CLINTON SP-LINC</td>
<td>LINCOLN-GOVFT SQ</td>
<td></td>
</tr>
<tr>
<td>TIME PERIOD</td>
<td>ON</td>
<td>OFF</td>
<td>TIME</td>
<td>ON</td>
<td>OFF</td>
</tr>
<tr>
<td>-------------</td>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>A-14 6-0-159</td>
<td>114</td>
<td>39</td>
<td>11.7</td>
<td>62</td>
<td>39</td>
</tr>
<tr>
<td>91 6-0-1159</td>
<td>132</td>
<td>60</td>
<td>11.1</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>131 6-0-17159</td>
<td>94</td>
<td>39</td>
<td>12.8</td>
<td>62</td>
<td>39</td>
</tr>
<tr>
<td>101 6-0-24159</td>
<td>44</td>
<td>28</td>
<td>9.9</td>
<td>62</td>
<td>28</td>
</tr>
<tr>
<td>61 6-0-24159</td>
<td>206</td>
<td>146</td>
<td>11.5</td>
<td>62</td>
<td>146</td>
</tr>
</tbody>
</table>

#### TIME PERIOD TOTALS

<table>
<thead>
<tr>
<th>TIME PERIOD</th>
<th>TOTAL ON</th>
<th>TOTAL OFF</th>
<th>AVE. RUN</th>
<th>NO. OF VEH.</th>
<th>VEH.- MILES</th>
<th>VEH.- HOURS</th>
<th>(PASS./PER.) VEH.- TRIP</th>
<th>(PASS./PER.) VEH.- MILE</th>
<th>(PASS./PER.) VEH.- HOURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM PEAK 6-0-159</td>
<td>495</td>
<td>495</td>
<td>49.5</td>
<td>2</td>
<td>80.4</td>
<td>46.8</td>
<td>6.3</td>
<td>4.4</td>
<td>68.4</td>
</tr>
<tr>
<td>MIDDAY 91 6-0-1159</td>
<td>947</td>
<td>947</td>
<td>94.7</td>
<td>14</td>
<td>146.0</td>
<td>11.5</td>
<td>60.5</td>
<td>5.7</td>
<td>4.6</td>
</tr>
<tr>
<td>PM PEAK 131 6-0-17159</td>
<td>947</td>
<td>947</td>
<td>94.7</td>
<td>7</td>
<td>74.2</td>
<td>6.1</td>
<td>97.8</td>
<td>6.4</td>
<td>4.9</td>
</tr>
<tr>
<td>MIDNIGHT 101 6-0-24159</td>
<td>834</td>
<td>834</td>
<td>83.4</td>
<td>7</td>
<td>73.5</td>
<td>4.7</td>
<td>33.4</td>
<td>4.2</td>
<td>2.5</td>
</tr>
<tr>
<td>ALL DAY 61 6-0-24159</td>
<td>2888</td>
<td>2888</td>
<td>288.8</td>
<td>38</td>
<td>381.3</td>
<td>28.9</td>
<td>65.6</td>
<td>5.3</td>
<td>4.2</td>
</tr>
</tbody>
</table>

**SOURCE:** GM
RUN SUMMARY

header + { { key + data } }
trip TPs

header = file-ID + DOW-table + TOD-table + weather-code + school-code

key = line + direction + day-int + (time-int) + (trip)

data = TP + shed-run-time + run-time-dev + (run-time-dev-shed-dev) + sample-size
<table>
<thead>
<tr>
<th>TIMEPOINT</th>
<th>TRIP</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALVARADO/PICO</td>
<td>1070</td>
<td>0610:130</td>
<td>+01:133</td>
<td></td>
<td>0513:130</td>
<td>0100</td>
<td></td>
<td>0612:130</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>MONTANA/LIBERTY</td>
<td>1080</td>
<td>0612:130</td>
<td>+01:145</td>
<td></td>
<td>0610:130</td>
<td>0100</td>
<td></td>
<td>0614:00</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>ALVARADO/PICO</td>
<td>1090</td>
<td>0713:330</td>
<td>-11:30</td>
<td></td>
<td>0812:030</td>
<td>+02:100</td>
<td></td>
<td>0814:100</td>
<td>-01:00</td>
<td></td>
</tr>
<tr>
<td>ALVARADO/6TH</td>
<td>1100</td>
<td>0814:300</td>
<td>-13:30</td>
<td></td>
<td>0812:300</td>
<td>+02:130</td>
<td></td>
<td>0815:300</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>Driver Number</td>
<td>6407</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DIRECTION NORTHBOUND**

**PREVIOUS RUN**

<table>
<thead>
<tr>
<th>TIMEPOINT</th>
<th>TRIP</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALVARADO/6TH</td>
<td>1150</td>
<td>1315:000</td>
<td>+01:145</td>
<td></td>
<td>1313:500</td>
<td>0100</td>
<td></td>
<td>1410:4100</td>
<td>0100</td>
<td></td>
</tr>
</tbody>
</table>

**DIRECTION SOUTHBOUND**

<table>
<thead>
<tr>
<th>TIMEPOINT</th>
<th>TRIP</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALVARADO/BEVERLY</td>
<td>1070</td>
<td>0612:610</td>
<td>-01:130</td>
<td></td>
<td>0610:510</td>
<td>0100</td>
<td></td>
<td>0614:100</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>ALVARADO/6TH</td>
<td>1080</td>
<td>0612:930</td>
<td>+02:100</td>
<td></td>
<td>0610:310</td>
<td>0100</td>
<td></td>
<td>0614:00</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>SANTA/FIGUERDA</td>
<td>1150</td>
<td>0910:910</td>
<td>+01:130</td>
<td></td>
<td>0810:410</td>
<td>-01:130</td>
<td></td>
<td>0813:00</td>
<td>0100</td>
<td></td>
</tr>
</tbody>
</table>

**FOLLOWING RUN**

<table>
<thead>
<tr>
<th>TIMEPOINT</th>
<th>TRIP</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
<th>SCHEDULE DEPARTURE</th>
<th>DEPART DEVIATION</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALVARADO/BEVERLY</td>
<td>1070</td>
<td>0612:610</td>
<td>-01:130</td>
<td></td>
<td>0610:510</td>
<td>0100</td>
<td></td>
<td>0614:100</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>ALVARADO/6TH</td>
<td>1080</td>
<td>0612:930</td>
<td>+02:100</td>
<td></td>
<td>0610:310</td>
<td>0100</td>
<td></td>
<td>0614:00</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>SANTA/FIGUERDA</td>
<td>1150</td>
<td>0910:910</td>
<td>+01:130</td>
<td></td>
<td>0810:410</td>
<td>-01:130</td>
<td></td>
<td>0813:00</td>
<td>0100</td>
<td></td>
</tr>
</tbody>
</table>

**SOURCE:** Gould, Inc.
<table>
<thead>
<tr>
<th>SCHEDULE DEVIATION</th>
<th>WILLIAMS &amp; WYOMING</th>
<th>READING &amp; SUMMIT</th>
<th>TIME POINTS</th>
<th>READING &amp; CALIFORNIA</th>
<th>HEADING &amp; CLINTON SPRINGS</th>
<th>READING &amp; LINCOLN</th>
<th>GOVERNMENT SQUARE OVERLOAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>M 10 TO 9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>T 9 TO 8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>M 8 TO 7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T 7 TO 6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E 6 TO 5</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>A 5 TO 4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>M 4 TO 3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>L 3 TO 2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>T 2 TO 1</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>1 TO 0</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>0 TO 1</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>M 1 TO 2</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>T 2 TO 3</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>M 3 TO 4</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>2 TO 5</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>L 5 TO 6</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>A 6 TO 7</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>T 7 TO 8</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>E 8 TO 9</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>9 TO 10</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>OVER 16</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

*DETONES ON TIME DEFINITION

SOURCE: GM
SCHEDULE DEVIATION SUMMARY

header + { { key + data } } trips TPs

header = file-ID + DOW-table + TOD-table

key = line + direction + day-int + time-int

t = +t_2

data = TP + { buses-in-sched-dev-int } +

t = -t_1

sample-size

NOTE: 1 minute sched. dev. intervals

\[
\begin{align*}
-t_1 &= -10 \text{ min.} \\
+t_2 &= +10 \text{ min.}
\end{align*}
\] 22 intervals
SECTION 15 SUMMARY

header + { key + data }

header = file-ID + DOW-table + TOD-table

key = line + direction + day-int + time-int

data = psgrs-boarded + bus-trip-distance + psgr-mi + bus-trip-time + psgr-min + capacity-mi + seat-mi + bus-trips

NOTE: DOW-table & TOD-table specified by UMTA

Weekdays, Sat., Sun.
AM-peak, Mid-day, PM-peak, Night
SIGNPOST-TABLE-1  (SIGNTB)  (AVM Log Tape)

x-coord + y-coord + distance +
east-code + line +
east-chain + west-chain +
north-chain + south-chain

NOTE:

- 1 entry per signpost (917)
- "chain" values tell how many records between this SP and the next one in each direction
- ordered by north-code, then east-code
NOTES:

- 255 entries—one for each possible north code. Their order defines a given north code.

- This table used as a key into SIGNTB.
SUMMARY TAPE

Header = ID + start-date + end-date
+ (Section-15-Summary)
+ (Run-Summary)
+ (Ride-Detailed-Summary)
+ (Ride-Trip-Summary)
+ (Sched-Dev-Summary)
+ (Overload-Summary)
+ (Layover-Summary)
TIME OF DAY (TOD) TABLE

table-length
TOD-code + table-length + { end-time }
i = 1

\[
\begin{align*}
\text{TOD-code} &= \left\{ \\
\text{ALL, if each trip is to be included separately} \\
\text{INT, if trips are to be aggregated over time intervals} \end{align*}
\]

A-23
TP-TRIP

header + { key + data } }
trips TPs

header = file-ID + date + day + weather + school

key = line + direction + trip

data = run + bus + TP + time + sched-dev +
psgrs-on-between-TPs + psgrs-off-bet-TPs +
psgrs-aboard + max-psgrs-bet-TPs +
max-psgr-bus-stop
TRACKED BUS (AVM Log Tape-Type 3 File)

block header + \{[bus-stop \ bus-assnment/pullout \ dispatcher-action]\}

block header = AVM-block-start-time + date + day + weather + {line + sched + AVM-config}

lines

bus-stop = time + line + run + bus + trip +
direction + origin + destination +
stop + TP + sched-dev + sched-adj +
shed-hywy + hdwy dev + psgr-aboard +
psgrs-boarding + psgrs-alighting +
total-boardings + total-alighting +
psgrs-on-leader + load-diff + run-time +
psgr-wait + psgr-mi + psgr-trip-time +
psgr-trip-delay + tactical-code +
trouble-report +

3

SPM-position +1 \{tactics\} + other-flags
## Table E: Trouble Report (CS-10)

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle</td>
<td>3108</td>
</tr>
<tr>
<td>Division</td>
<td>99</td>
</tr>
<tr>
<td>Call (Location)</td>
<td>WILSH/WESTERN STATION</td>
</tr>
<tr>
<td>Status</td>
<td>XX</td>
</tr>
<tr>
<td>Time</td>
<td>1159A</td>
</tr>
<tr>
<td>Operator ID</td>
<td>XXXX</td>
</tr>
<tr>
<td>Line/Run</td>
<td>17-12</td>
</tr>
<tr>
<td>Location</td>
<td>XXXXXXXXXX</td>
</tr>
<tr>
<td>Time</td>
<td>XXX</td>
</tr>
<tr>
<td>CLRD</td>
<td>XXXX</td>
</tr>
<tr>
<td>Accident</td>
<td>EMERGENCY</td>
</tr>
<tr>
<td>Operations</td>
<td>MAINTENANCE PROBLEMS</td>
</tr>
<tr>
<td># Injuries</td>
<td>SAS ALARM</td>
</tr>
<tr>
<td>Event at Scene</td>
<td>ASSAULT-OFFER</td>
</tr>
<tr>
<td>Referred Aid</td>
<td>ASSAULT-PASSenger</td>
</tr>
<tr>
<td>To Hospital</td>
<td>ROBBERY</td>
</tr>
<tr>
<td>Ambulance</td>
<td>OUT LATE</td>
</tr>
<tr>
<td>Which Hospital</td>
<td>MISCELLANEOUS</td>
</tr>
<tr>
<td>WITNESSES</td>
<td>ALTERNATION</td>
</tr>
<tr>
<td>PHOTOS</td>
<td>DISTURBANCE</td>
</tr>
<tr>
<td>BUS MOVING</td>
<td>INTOX-PASSenger</td>
</tr>
<tr>
<td>BUS DAMAGE</td>
<td>LOST ARTICLE</td>
</tr>
<tr>
<td>Mispellings</td>
<td>REFERRED AID</td>
</tr>
<tr>
<td>Service Delay</td>
<td>XXXX</td>
</tr>
<tr>
<td>New Bus No.</td>
<td>XXXX</td>
</tr>
<tr>
<td>Location</td>
<td>XXXXXXXXXX</td>
</tr>
<tr>
<td>Engines Replacement</td>
<td>XXXX</td>
</tr>
<tr>
<td>Time</td>
<td>XXXX</td>
</tr>
<tr>
<td>Copies</td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td>COULD, Inc.</td>
</tr>
</tbody>
</table>

---

**Summary**

- Reported By: XXXX
- To Time: XXXX

---

**Lines 23-40**

---

**Lines 42-52**

---

**Service Delay:** XXXX

**New Bus No:** XXXX

**Location:** XXXXXXXXXX

**Direction:** XXXX

**Time:** XXXX
TROUBLE REPORT (AVM Log Tape-Type 2 File)

\[ \text{block-pointer} = \{ \text{block-number-of-} \text{th} \text{-trouble} \]
\[ i=1 \]
\[ \text{report} \} \]

\[ \text{data} = \text{trouble-report-text} \]
VEHICLE-LOCATION-AND-STATUS (CLST) (AVM Log Tape)

VLST 1 = vehicle + line + run + bus-type

VLST 2 = for-internal-use-only

6 {schedule-deviation-thresholds} +
1 {computed-sched-deviation +

VLST 3 =

1 {display-sched-thresholds} 
3

VLST 4 = bus + line + run + veh-ID + bus-type + status

NOTES:

• Random route + 41 + 44 + 89 + 83

• 1 entry per vehicle (220 total)

• vehicle-ID not the same as vehicle-number
The processes required to form the files defined in Appendix A are detailed here. The numbering system corresponds to that used in Section 2. For example, B.1.3.1, "Form FSTRIP Files" is described in Section 2.1.3.1 and refers to the process with the same name, numbered 1.3.1, shown in the "balloon chart" of Figure 2-5.
B.1 Form AVM Files.

Pertinent logical procedures comprising this major module are described below.

B.1.1 Gould Fortran Programs.

As this module has been separately developed and documented by Gould, details of the logical processes are not given here.

B.1.2 Select and Sort.

Do While records exist

Read tracked-bus Into temp.
Select (record-type):

Case(block-header):
If this is first block-header read Then
Write bus-stop-header From temp.

Case(bus-stop-record):
Write bus-stop-by-line(line) From temp.

Endselect.

Enddo.

For all lines
Sort bus-stop-by-line(line) By direction, trip, time.

Concatenate bus-stop-header with
bus-stop-by-line(all-lines) to form ordered-bus-stop file.

B.1.3 Update perf tapes.

The detailed processes comprising this module are described below.
B.1.3.1 Form TP-Trip File

Program Variables:

header
id
date
day
school
weather

new

key

line

direction

trip

data1
time
run
bus
TP

adj-sched-dev
psgrs-aboard

data2

stop
sched-adjust
psgrs-boarding
psgrs-alighting
total-on
total-off

prev: same as above

out

key
data1
calc

sched-dev
psgrs-on
psgrs-off
max-psgrs
max-psgrs-stop

save

total-on * to compute total psgrs *
total-off * on/off between TPs *

max

max-psgrs
max-psgrs-stop

B-3
Main:

Call process-header.
Call initialize.
Call process-data.

Process-Header:

Read ordered-bus-stop Into header.
id = TP-trip-id.
Write TP-trip From header.

Initialize:

Read ordered-bus-stop into new
at end Call error1.
prev = new.
Call trip-begin.
save = prev By-Name.

Process-Data:

Do While new-key ≠ high-values

Do While new-key = prev-key

prev = new.
Call stop-update.
If prev-TP ≠ 0
Call TP-update.
Read ordered-bus-stop into new
at end new-key = high-values.

Call trip-end.
Call trip-begin.
Stop-Update:

If prev-psgrs-aboard > max-psgrs
    max-psgrs = prev-psgrs-aboard
    max-psgr-stop = prev-stop.

TP-Update:

    psgrs-on = prev-total-on minus save-total-on.
    psgrs-off = prev-total-off minus save-total-off.
* Compute psgrs aboard when bus leaves stop *
    sched-dev = adj-sched-dev minus sched-adj.
    psgrs-aboard = psgrs-aboard + psgrs-boarding minus
                 psgrs-alighting.
    out = prev By-Name.
    calc = max By-Name.
    save = prev By-Name.
    Write TP-trip From out.
    max = 0.

* First and last bus stop data saved (for trip, layover times), even if they're not time points *

Trip-Begin:

    out = 0.
    max = 0.
    out = prev By-Name.
    out-TP = begin-code.
    Write TP-trip From out.

Trip-End:

    out = prev By-Name.
    out-TF = end-code.
    Write TP-trip From out.
    If new-key => high-values
        prev = new
        Read ordered-tus-stop Into new.
B.1.3.2 Form New Summary Tape.

The detailed processes comprising this module are described below.

B.1.3.2.1 Information common to Aggregate programs.

Input Variables (from TP-trip file):

new
  key
  line
  direction
  trip
  data1
    run
    bus
    TP
  data2
    time
    sched-adj
    adj-sched-dev
  data3
    psgrs-on
    psgrs-off
    psgrs-aboard
    max-psgrs
    max-psgrs-stop
prev: same as new

Header variables:

header1
  id
  begin-date
  end-date
  DOW-code
  TGD-code
  weather-code
  school-code

header2
  id
  date
  day
  weather
  school

Process-Header:

If **-summary exists Read summary into header1.
  First day of new summary tape *
  Else Read selection-criteria into header1.
  Read TP-trip into header2.
  If weather-code = all or weather = weather-code
    If school-code = all or school = school-code
      header2-id = **-aggregate-id
      Write **-aggregate from header2
      Return.
      * Continue processing *
  Else terminate processing.
      * Selection criteria not *
      * satisfied by today's *
      * conditions *

B-6
Time-Int-Table(new-data2,new-key):

* Determine which time interval a given trip start time is in;
  the trip-table gives a "phantom start time" for each bus
  that does not start its trip at the beginning of the line *

  time-check = trip-table(new-key).

* If new-trip is in table Then
  time-check = phantom-start-time.
Else time-check = 0. *

  If time-check = 0 Then
    sched-time = new-time + sched-dev
    time-int = TOD-table(sched-time).
  Else time-int = TOD-table(time-check).

* TOD-table chooses the proper TCD-interval based on the
  originally-chosen aggregation intervals *
B.1.3.2.1.1 Form Daily-Section-15-Aggregate File

Program Variables:

cut
key
line
direction
day-int
time-int
data
psgrs-boarded
bus-miles
psgr-miles
bus-mins
psgr-mins
capacity-miles
seat-miles
bus-trips * sample size *

calc(4)
data (same as above)

* calc(i,j)
* i = 1 - time interval or trip aggregation
* 2 - direction aggregation
* 3 - line aggregation
* 4 - system aggregation

Main:

Call process-header.
Call initialize.
Call process-data.

Initialize:

calc = 0.
Read TP-trip Into prev At end Call err1.
Read TP-trip Into new At end Call err2.
Call aggregate-begin.
Process-Data:

Do While new-key != high-values
  Do While new-key = prev-key
    Call compute.
    prev = new.
    Read TP-trip Into new At end new-key = high values.
  End.
Call trip-end.
Call system-control-break.
End.

Aggregate-Begin:

* Initialize out-key data *
  out = 0.
  out-day-int = day-table(header-day).
  out-time-int = time-int-table(new-data2,new-key).
  out-line = new-line.
  out-direction = new-direction.

Compute:

calc-psgrs-boarded(*) = calc-psgrs-boarded(*) + psgrs-on(new-TP).
avg-psgrs = (psgrs-aboard(prev-TP) + psgrs-aboard(new-TP)) / 2.
miles = distance-table(prev-TP,new-TP).
mins = new-time minus prev-time.
calc-bus-mins(*) = calc-bus-mins(*) + mins.
calc-bus-miles(*) = calc-bus-miles(*) + miles.
calc-psgr-miles(*) = calc-psgr-miles(*) + (avg-psgrs x miles).
calc-psgr-mins(*) = calc-psgr-mins(*) + (avg-psgrs x mins).
seats = seat-table(bus).
capacity = seats.
calc-capacity-miles(*) = calc-capacity-miles(*) + (capacity x miles).
calc-seat-miles(*) = calc-seat-miles(*) + (seats x miles).
Trip-End:

* When new record is from different trip, line or direction, * perform necessary summaries
  * calc-bus-trips(*) = calc-bus-trips(*) + 1.
  * time-int = time-int-table(new-data2,new-key).
  * If new-line != prev-line
    Call line-control-break.
  * Else If new-direction != prev-direction
    Call direction-control-break.
  * Else If time-int != out-time-int
    Call time-control-break.
  * Otherwise, perform no additional processing *

* Have first record in new trip--read another *
  * If new-key == high-values
    Call aggregate-begin
    prev = new
    Read TP-trip Into new At end Call err3.

Line-Control-Break:
  Call direction-control-break.
  out-direction = line-agg-code.
  Call output(3).

Direction-Control-Break:
  Call time-control-break.
  out-time-int = dir-agg-time-code.
  Call output(2).

Time-Control-Break:
  Call output(1).

System-Control-Break:
  out-line = system-agg-code.
  Call output(4).

Output(i):

* Write output *
  * out = calc(i) By-Name.
  Write Section-15-aggregate From out.
  calc(i) = 0.
B.1.3.2.1.2 Form Daily-Run-Aggregate File

Program Variables:

    * layover output format *
    * same as input *

    out
    key
    line
    direction
    day-int
    (time-int)
    (trip)
    data
    TP
    sched-run-time
    run-time-dev
    adj-run-time-dev
    run-time-dev-var
    sample-size

    calc(2, trip-agg-code)
    data (same as above)

    * calc(i, j)
    * i = 1 - time interval or trip aggregation
    * 2 - direction aggregation

    save
    time
    sched-dev
    adj-sched-dev

Main:

Call process-header.
Call initialize.
Call process-data.

Initialize:

    calc = 0.
    Read TP-trip Into prev At end Call err1.
    Read TP-trip Into new At end Call err2.
    Call aggregate-begin.
Process-Data:

Do While new-key ^= high-values
  Do While new-key = prev-key
    Call compute.
    prev = new.
    Read TP-trip Into new At end new-key = high values.
    Call trip-end.
  End.
End.

Aggregate-Begin:

* Initialize out-key data *
  out = 0.
  out-day-int = day-table(header-day).
  If TOO-code ^= all
    out-time-int = time-int-table(new-data2,new-key).
  Else out-trip = new-trip.
  out-line = new-line.
  out-direction = new-direction.

Compute:

* Save 'first bus stop' data from trip totals *
  If prev-TP = begin-code
    save = prev By-Name
    Write layover from prev.

* Check for 'last bus stop' *
  If new-TP = end-code
    Write layover From new
    Return.

* Compute times and add to previously saved ones *
  run-time = new-time minus prev-time.
  sched-time = (new-time + new-sched-dev) minus
               (prev-time + prev-sched-dev).
  adj-sched-time = (new-time + new-adj-sched-dev) minus
                   (prev-time + prev-adj-sched-dev).
  dev = sched-time minus run-time.
  adj-dev = adj-sched-time minus run-time.
  calc-sched-run-time(*,new-TP) = calc-sched-run-time(*,new-TP) +
                                  sched-time.
  calc-run-time-dev(*,new-TP) = calc-run-time-dev(*,new-TP) + dev.
  calc-adj-run-time-dev(*,new-TP) = calc-adj-run-time-dev(*,new-TP) +
                                    adj-dev.
  calc-run-time-dev-var(*,new-TP) = calc-run-time-dev-var(*,new-TP) +
                                   dev*2.
  calc-sample-size(*,new-TP) = calc-sample-size(*,new-TP) + 1.
Trip-End:

* When new record is from different trip, line or direction, *
  * perform necessary summaries *

run-time = prev-time minus save-time.
sched-time = (prev-time + prev-sched-dev) minus
  (save-time + save-sched-dev).
adj-sched-time = (prev-time + prev-adj-sched-dev) minus
  (save-time + save-adj-sched-dev).
dev = sched-time minus run-time.
adj-dev = adj-sched-time minus run-time.
calc-sched-run-time(*,trip-agg-code) =
  calc-sched-run-time(*,trip-agg-code) + sched-time.
calc-run-time-dev(*,trip-agg-code) =
  calc-run-time-dev(*,trip-agg-code) + dev.
calc-adj-run-time-dev(*,trip-agg-code) =
  calc-adj-run-time-dev(*,trip-agg-code) + adj-dev.
calc-run-time-dev-var(*,trip-agg-code) =
  calc-run-time-dev-var(*,trip-agg-code) + dev**2.
calc-sample-size(*,trip-agg-code) =
  calc-sample-size(*,trip-agg-code) + 1.
time-int = time-int-table(nev-data2,new-key).

If new-line != prev-line or new-direction != prev-direction
  Call direction-control-break.
Else If time-int != out-time-int or
  TOD-code = all
  Call time-control-break.
* Otherwise, perform no further processing  *

* Have first record in new trip--read another *
  If new-key != high-values
    prev = new
    Call aggregate-begin
    Read TP-trip Into new At end Call err3.

Direction-Control-Break:
  Call time-control-break.
  If TOD-code = all out-trip = dir-agg-trip-code.
  Else out-time-int = dir-agg-time-code.
  Call output(2).

Time-Control-Break:
  Call output(1).
Output(i):

* Form averages and output *
For j = 1 to trip-agg-code
  If calc-sample-size(i,j) > 0
    calc-sched-run-time(i,j) = calc-sched-run-time(i,j) /
    calc-sample-size(i,j)
    calc-run-time-dev-var(i,j) = (calc-run-time-dev-var(i,j)
      minus calc-run-time-dev(i,j)**2) / calc-sample-size(i,j)
    calc-run-time-dev(i,j) = calc-run-time-dev(i,j) /
    calc-sample-size(i,j)
    out = calc(i,j) By-Name
  Write run-aggregate from out.
  calc(i,*j) = 0.
B.1.3.2.1.3 Form Daily-Ride-Aggregate Files

Program Variables:

<table>
<thead>
<tr>
<th>out1 key</th>
<th>out2 key</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>line</td>
<td>psgr-trips</td>
<td></td>
</tr>
<tr>
<td>direction</td>
<td>psgr-miles</td>
<td></td>
</tr>
<tr>
<td>day-int</td>
<td>load-factor</td>
<td></td>
</tr>
<tr>
<td>(time-int)</td>
<td>standee-mins</td>
<td></td>
</tr>
<tr>
<td>(trip)</td>
<td>st-min-per-st</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>bus-trips * sample-size*</td>
<td></td>
</tr>
<tr>
<td>TP</td>
<td>calc1(2,trip-agg-code)</td>
<td></td>
</tr>
<tr>
<td>psgrs-on</td>
<td>calc2(4)</td>
<td></td>
</tr>
<tr>
<td>psgrs-off</td>
<td>data (same as above)</td>
<td></td>
</tr>
<tr>
<td>max-psgrs</td>
<td>save</td>
<td></td>
</tr>
<tr>
<td>max-psgr-stop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>standees</td>
<td></td>
<td></td>
</tr>
<tr>
<td>standee-time(3)</td>
<td>load</td>
<td></td>
</tr>
<tr>
<td>mins-wi-standees</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sample-size</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

calc1(i,j) *

* i = 1 - time interval or trip aggregation *
* 2 - direction aggregation *
* 3 - line aggregation *
* 4 - system aggregation *

Main:

Call process-header.
Call initialize.
Call process-data.

Initialize:

calc1, calc2, save = 0.
Read TP-trip into prev at end call err1.
Read TP-trip into new at end call err2.
Call aggregate-begin.
Process-Data:

Do While new-key != high-values
  Do While new-key = prev-key
    Call compute.
    prev = new.
    Read TP-trip Into new At end new-key = high values.
  End.
Call trip-end.
Call system-ctrl-break.
End.

Aggregate-Begin:

seats = seat-table(bus).
st-min2, min2 = 0.
* Initialize out-key data
  out, out1, out2 = 0.
out-day-int = day-table(header-day).
If TCC-code = all
  out-time-int = time-int-table(new-data2,new-key).
Else out-trip = new-trip.
out-line = new-line.
out-direction = new-direction.
Compute:

* Detailed Ridership Computations *
  calc1-psgrs-on(*,new-TP) = calc1-psgrs-cn(*,new-TP) + new-psgrs-on.
  calc1-psgrs-off(*,new-TP) = calc1-psgrs-off(*,new-TP) + new-psgrs-off.
  If new-max-psgrs > calc1-max-psgrs(*,new-TP)
    calc1-max-psgrs(*,new-TP) = new-max-psgrs
    calc1-max-stop(*,new-TP) = new-max-stop.
  calc1-sample-size(*,new-TP) = calc1-sample-size(*,new-TP) + 1.

* Trip Ridership Computations (others at end) *
  miles = distance-table(prev-TP,new-TP).
  mins = new-time minus prev-time.
  avg-psgrs = (prev-psgrs-aboard + new-psgrs-aboard) / 2.
  save-load(*) = save-load(*) + avg-psgrs.
  calc2-psgr-miles(*) = calc2-psgr-miles(*) + (avg-psgrs x miles).
  calc2-psgr-mins(*) = calc2-psgr-mins(*) + (avg-psgrs x mins).
  calc2-psgr-trips(*) = calc2-psgr-trips(*) + psgrs-on.

* Compute Standee Data (trip-agg-code saves trip sums) *
  If (prev-psgrs-aboard or new-psgrs-aboard) > seats
    Call compute-standees
  calc1-standees(*,new-TP) = calc1-standees(*,new-TP) + standees
  * Trip standees calculated at control break *
    calc1-mins-wi-standees(*,new-TP) = calc1-mins-wi-standees(*,new-TP)
    mins-wi-standees
    calc1-mins-wi-standees(*,trip-agg-code) =
      calc1-mins-wi-standees(*,trip-agg-code) + mins-wi-standees
    calc2-standee-mins(*) = calc2-standee-mins(*) + mins-wi-standees

* Accumulate passenger-minutes-per-standee data *
  st-min2 = st-min2 + standee-mins
  min2 = min2 + mins-wi-standees
  If new-psgrs-aboard < seats
    calc2-standees(*) = calc2-standees(*) + (st-min2 / min2)
  st-min2, min2 =0
  Endif.

* Separate standee-mins according to number of standees *
  If standee-mins < 5
    calc1-standee-time(*,new-TP,1) = calc1-standee-time(*,new-TP,1) + standee-mins
    calc1-standee-time(*,trip-agg-code,1) =
      calc1-standee-time(*,trip-agg-code,1) + standee-mins.
  Else If standees > 10
    calc1-standee-time(*,new-TP,3) =
    calc1-standee-time(*,new-TP,3) + standee-mins
    calc1-standee-time(*,trip-agg-code,3) =
      calc1-standee-time(*,trip-agg-code,3) + standee-mins.
  Else calc1-standee-time(*,new-TP,2) =
    calc1-standee-time(*,new-TP,2) + standee-mins
    calc1-standee-time(*,trip-agg-code,2) =
      calc1-standee-time(*,trip-agg-code,2) + standee-mins.
  Endif.
Compute-Standees:

\[ p_{\text{min}} = \min(\text{prev-psgrs-aboard}, \text{new-psgrs-aboard}) \]
\[ p_{\text{max}} = \max(\text{prev-psgrs-aboard}, \text{new-psgrs-aboard}) \]
\[ \text{psgr-diff} = p_{\text{max}} - p_{\text{min}} \]
\[ \text{time-diff} = \text{new-time} - \text{prev-time} \]

\[ \text{If} \ (p_{\text{min}} \text{ and } p_{\text{max}}) > \text{seats} \]
\[ \text{time-over} = \text{time-diff} \]
\[ \text{standee-mins} = \left( \frac{\text{psgr-diff}}{2} \right) + (p_{\text{min}} - \text{seats}) \times \text{time-over} \]
\[ \text{Else} \]
\[ \text{excess-psgrs} = p_{\text{max}} - \text{seats} \]
\[ \text{ratio-over} = \frac{\text{excess-psgrs}}{\text{psgr-diff}} \]
\[ \text{time-over} = \text{time-diff} \times \text{ratio-over} \]
\[ \text{standee-mins} = \left( \frac{\text{excess-psgrs} \times \text{time-over}}{2} \right) \]

\[ \text{If} \ \text{time-diff} > 0 \]
\[ \text{standees} = \frac{\text{standee-mins}}{\text{time-diff}} \]
\[ \text{Else} \text{standees} = 0 \]

\[ \text{mins-wi-standees} = \text{time-over} \]

Trip-End:

* When new record is from different trip, line or direction, perform necessary summaries *
\[ \text{calc2-bus-trips(*)} = \text{calc2-bus-trips(*)} + 1 \]
\[ \text{time-int} = \text{time-int-table(new-data2,new-key)} \]
\[ \text{If} \ \text{new-line} = \text{prev-line} \]
\[ \text{Call line-control-break} \]
\[ \text{Else If} \ \text{new-direction} = \text{prev-direction} \]
\[ \text{Call direction-control-break} \]
\[ \text{Else If} \ \text{time-int} = \text{out-time-int} \text{or} \]
\[ \text{time-code} = \text{all} \]
\[ \text{Call time-control-break} \]
* Otherwise, perform no further processing *

* Have first record in new trip--read another *
\[ \text{If} \ \text{new-key} = \text{high-values} \]
\[ \text{Call aggregate-begin} \]
\[ \text{prev} = \text{new} \]
\[ \text{Read TP-trip Into new At end Call err3} \]

Line-Control-Break:
\[ \text{Call direction-control-break} \]
\[ \text{out-direction} = \text{line-agg-code} \]
\[ \text{Call output(3)} \]
Direction-Control-Break:
Call time-control-break.
If TOD-code = all out-trip = dir-agg-trip-code.
Else out-time-int = dir-agg-time-code.
Call output(2).

Time-Control-Break:
Call output(1).
save = 0.

System-Control-Read:
out-line = system-agg-code.
Call output(4).

Output(i):
* Write out data *
* Output-2 data *
avg-load = save-load(i) / calc2-sample-size(i).
calc2-load-factor(i) = avg-load / seats.
If calc2-standees(i) > 0 Then
  calc2-st-min-per-st(i) = calc2-standee-mins(i) / calc2-standees(i).
out2 = calc2 By-Name.
calc1-standees(i,trip-agg-code) = calc2-standees(i).
Write ride-trip-aggregate from out2.
calc2(i) = 0.
* Output-1 data *
If i <= 2
  * for ride-detailed file, only output for *
  * time & direction control breaks (line *
  * and system totals meaningless) *
  For j = 1 to trip-agg-code
    If calc1-sample-size(i,j) > 0 Then
      out1 = calc1(i,j) By-Name
      Write ride-detail-aggregate from out.
      calc1(i,*) = 0.
B.1.3.2.1.4 Form Daily-Sched-Dev-Aggregate File

Program Variables:

out
key
line
direction
day-int
time-int
data
TP
hist(22)
* calc-hist(i,j,k) *
sample-size
calc(2,trip-agg-code)
data (same as above) * note: there is a *
* calc-hist(i,j,k) *

* calc(i,j) *
* i = 1 - time interval or trip aggregation *
* 2 - direction aggregation *
* j = TP *
* k = histogram interval *

Main:
Call process-header.
Call initialize.
Call process-data.

Initialize:
calc = 0.
Read TP-trip Into prev At end Call err1.
Read TP-trip Into new At end Call err2.
Call aggregate-begin.

Process-Data:
Do While new-key ~= high-values
  Do While new-key = prev-key
    Call compute.
    prev = new.
    Read TP-trip Into new At end new-key = high values.
    Call trip-end.
End.
Aggregate-Begin:

* Initialize out-key data
  out = 0.
  out-day-int = day-table(header-day).
  out-time-int = time-int-table(new-data2,new-key).
  out-line = new-line.
  out-direction = new-direction.

Compute:

  calc-sample-size(*,new-TP) = calc-sample-size(*,new-TP) + 1.

  * Find histogram interval for sched-dev *
  If new-adj-sched-dev < -10
    calc-hist(*,new-TP,1) = calc-hist(*,new-TP,1) + 1.
  Else If new-adj-sched-dev > +10
    calc-hist(*,new-TP,22) = calc-hist(*,new-TP,22) + 1.
  Else index = integer(new-adj-sched-dev) + 12
    calc-hist(*,new-TP,index) = calc-hist(*,new-TP,index) + 1.

Trip-End:

* When new record is from different trip, line or direction, *
  * perform necessary summaries
  time-int = time-int-table(new-data2,new-key).
  If new-line != prev-line or new-direction != prev-direction
    Call direction-control-break.
  Else If time-int != out-time-int
    Call time-control-break.
  * Otherwise, perform no processing *

* Have first record in new trip--read another *
  If new-key != high-values
    Call aggregate-begin
    prev = new
    Read TP-trip Into new At end Call err3.

Direction-Control-Break:
  Call time-control-break.
  out-time-int = dir-agg-time-cd.
  Call output(2).

Time-Control-Break:
  Call output(1).
Output (i):

For j = 1 to trip-agg-code
    If calc-sample-size(i, j) > 0
        calc-hist(i, j, *) = calc-hist(i, j, *) / calc-sample-size
        out = calc(i, j) By-Name
        Write sched-dev-aggregate from out.
    calc(i, *) = 0.
B.1.3.2.1.5 Form Daily-Overload-Aggregate File

Program Variables:

out
  key
  line
  direction
  day-int
  time-int
  data
    TP
    hist{10}
    sample-size

  calc(2,trip-agg-code)
    data (same as above)

* calc(i,j)
  * i = 1 - time interval or trip aggregation
  * 2 - direction aggregation

Main:

  Call process-header.
  Call initialize.
  Call process-data.

Initialize:

  calc = 0.
  Read TP-trip Into prev At end Call err1.
  Read TP-trip Into new At end Call err2.
  Call aggregate-begin.

Process-Data:

  Do While new-key ^= high-values
    Do While new-key = prev-key
      Call compute.
      prev = new.
      Read TP-trip Into new At end new-key = high values.
    End
  End

End.
Aggregate-Begin:

* Initialize out-key data
  * out = 0.
  * capacity = seat-table!bus].
  * If time-code = all
    * out-time-int = time-int-table(new-data2,new-key).
  * Else out-trip = new-trip.
  * out-day = day-table(header-day).
  * out-line = new-line.
  * out-direction = new-direction.

Compute:

* Find load factor between time points
  * avg-load = (prev-psgrs-aboard +
    new-psgrs-aboard) / 2.
  * load-factor = avg-load / capacity.
  * calc-sample-size(*,new-TP) = calc-sample-size(*,new-TP) + 1.

* Find histogram interval
  * index = load-factor-table(load-factor).
  * calc-hist(*,new-TP,index) = calc-hist(*,new-TP,index) + 1.

Trip-Fnd:

* When new record is from different trip, line or direction, output averages
  * If new-line ≠ prev-line or new-direction ≠ prev-direction
    * Call direction-control-break.
  * Else Call time-control-break.

* Have first record in new trip—read another
  * If new-key ≠ high-values
    * Call aggregate-begin
    * prev = new
    * Read TP-trip Into new At end Call err3.

Direction-Control-Break:

  * Call time-control-break.
  * out-time-int = dir-agg-time-code.
  * Call output(2).

Time-Control-Break:

  * Call output(1).
Output(i):

* Write output *
  If calc-sample-size(i) > 0
    calc-hist(i,j,*) = calc-hist(i,j,*) / calc-sample-size(i,j)
    out = calc(i,j) By-Name
    Write overload-aggregate from out.
    calc(i,*) = 0.
B.1.3.2.1.6 Form Daily-Layover-Aggregate File

Program Variables:

Modification of usual input record definition:

new
key
line
direction
run
trip
data
bus
TP
-
-
-
prev (same as above)

out
key
line
direction
day-int
(time-int)
(trip)
data
sched-layover
layover-dev
sample-size
calc(2)
data (same as above)

* calc(i)
* i = 1 - time interval or trip aggregation
* 2 - direction aggregation

Main:

Call process-header.
Call initialize.
Call process-data.

Initialize:

calc = 0.
Read TP-trip Into prev At end Call err1.
Read TP-trip Into new At end Call err2.
Call aggregate-begin.
Process-Data:

\[ \text{Do While new-key } \neq \text{ high-values} \]
\[ \text{Do While new-key } = \text{ prev-key} \]
\[ \text{prev = new.} \]
\[ \text{Read TP-trip Into new At end new-key } = \text{ high values.} \]
\[ \text{Call trip-end.} \]
\[ \text{End.} \]

Aggregate-Begin:

* Initialize out-key data *
out = 0.
out-day-int = day-table(header-day).
If TCD-code, = all
out-time-int = time-int-table(new-data2,new-key).
Else out-trip = new-trip.
out-line = new-line.
out-direction = new-direction.

Trip-End:

* When new record is from different trip, line, run or direction, perform necessary summaries *
If new-run = prev-run
layover = new-time minus prev-time
calc-sched-layover(*) = calc-sched-layover(*) +
((new-time + new-adj-sched-dev) minus
(prev-time + prev-adj-sched-dev))
calc-layover-dev(*) = calc-layover-dev(*) +
(layover minus calc-sched-layover(*))
calc-sample-size(*) = calc-sample-size(*) + 1.
time-int = time-int-table(new-data2,new-key).
If new-line, = prev-line or new-direction, = prev-direction
Call direction-control-break.
Else If time-int, = out-time-int or
new-run, = prev-run or
TCD-code, = all
Call time-control-break.
* Otherwise, perform no further processing *

* Have first record in new trip--read another *
If new-key, = high-values
Call aggregate-begin
prev = new
Read TP-trip Into new At end Call err3.
Direction-Control-Freak:
  Call time-control-break.
  If TOD-code = all out-trip = dir-agg-trip-code.
  Else out-time-int = dir-agg-time-code.
  Call output(2).

Time-Control-Freak:
  Call output(1).

Output(i):

  * Form averages and output *
  If calc-sample-size(i) > 0
    calc-sched-layover(i) = calc-sched-layover(i) / sample-size(i)
    calc-layover-dev(i) = calc-layover-dev(i) / sample-size(i)
    out = calc(i) By-Name.
    Write layover-aggregate from out.
    calc(i) = 0.
B.1.3.2.2 Update

The two input files (aggregate and old-summary) and the output file (new-summary) have the same format.

Main:

Call process-header.
Call initialize.
Call process-data.

Initialize:

Read aggregate Into aggregate At end Call err1.
Read old-summary Into summary At end Call err2.

Process-Data:

Do While aggregate-key $\neq$ high-values and summary-key $\neq$ high-values

If summary-key $<$ aggregate-key
Write new-summary From summary
Read old-summary Into summary
At end summary-key = high-values.

Else If summary-key = aggregate-key
Call update
Read aggregate Into aggregate
At end aggregate-key = high-values.

Else Write new-summary From aggregate
Read aggregate Into aggregate
At end aggregate-key = high-values.

Update:

For $i = 1$ to max-summary-elements

If summary-element(i) is a sum
summary-element(i) = summary-element(i) + agg-element(i).

Else If summary-element(i) is an average or percentage
old-size = summary-sample-size(i)
old-avg = summary-avg(i)
Call average(old-size, old-avg, agg-sample-size(i), agg-avg(i), summary-sample-size(i), summary-avg(i))
new-avg = summary-avg(i).
When a variance is updated, previous step will have updated the average (new) pertaining to that variable.

Else If summary-element(i) is a variance
   old-var = summary-var(i)
   Call Variance(old-size,old-avg,old-var,
                agg-sample-size(i),agg-avg(i),agg-var(i),
                summary-sample-size(i),new-avg,
                summary-var(i)).
Else If summary-element(i) is an extreme
   If agg-element(i) is more-extreme-than
      summary-element(i)
      summary-element(i) = agg-element(i).

Average(old-size,old-avg,addl-size,addl-avg,new-size,new-avg):
* Method to update an existing average (old) when another average (addl) becomes available *

   new-size = old-size + addl-size.
   If new-size > 0 Then
      x12 = old-avg minus addl-avg
      n23 = addl-size / new-size
      new-avg = old-avg minus (n23 x x12).
   Else Call error-handling.

Variance(old-size,old-avg,old-var,addl-size,addl-avg,addl-var,
         new-size,new-avg,new-var):
* Method to update an existing variance (old) when another (addl) becomes available *

   new-size = old-size + addl-size.
   If new-size > 0 Then
      x13sq = old-avg**2 minus new-avg**2
      x12sq = old-avg**2 minus addl-avg**2
      v12 = old-var minus addl-var
      n23 = addl-size / new-size
      new-var = old-var + x13sq minus (n23 x (v12 + x12sq)).
   Else Call error-handling.
B.1.3.3 Form New-Perf-Data-Base Tape.

Read old-perf-data-base header.
Read TP-trip header.

Write new-perf-data-base header.

Do While old-perf-data-base records exist
    Read old-perf-data-base Into temp.
    Write new-perf-data-base From temp.
Enddo.

Do While TP-trip records exist
    Read TP-trip Into temp.
    Write new-perf-data-base From temp.
Enddo.
B.1.3.4 Form Daily Exception Reports.

The detailed processes comprising this module are described below.

B.1.3.4.1 Form Daily Schedule Deviation Report

Program Variables:

- \text{save(3,trip-agg-code)}
- \text{TP}
- \text{time}
- \text{adj-sched-dev}
- \text{passengers-aboard}

\text{limit}
- \text{early}
- \text{late}

Main:

- \text{Call process-header.}
- \text{Call initialize.}
- \text{Call process-data.}

Initialize:

- \text{Read selection-criteria into limit.}
- \text{Read TP-trip into new}
  - at end Call error1.
  - prev = new.
  - Call read-three-trips.
  - If new-key = high-values Call error2.
  - Call first-trip.

Process-Data:

- \text{Do While new-key -= high-values}

  * Trip changes handled in read-trip *

  - \text{Do While new-line = prev-line}
    - and new-direction = prev-direction

    - Call check-tolerances.
    - \text{save(1,*)} = \text{save(2,*)}.
    - \text{save(2,*)} = \text{save(3,*)}.
    - prev = new.
    - Call read-trip(3).

- \text{Call last-trip.}
Read-Three-Trips:

\[
\text{limit-count} = 0.
\]

\[
\text{save} = 0.
\]

Call read-trip(1). * assumes at least 3 trips in *
Call read-trip(2). * a given line & direction *
Call read-trip(3).

Read-Trip(i):

\[
\text{Do While new-key = prev-key}
\]

\[
\text{prev = new.}
\]

\[
\text{If prev-TP \neq \text{begin-code} and prev-TP \neq \text{end-code}}
\]

\[
\text{save(i, prev-TP) = prev By-Name.}
\]

\[
\text{Read TF-trip Into new}
\]

\[
\text{at end new-key = high-values.}
\]

Check-Tolerances:

\[
\text{For } i = 1 \text{ to trip-agg-code}
\]

\[
\text{If save-sched-dev(2,i) < early or save-sched-dev(2,i) > late}
\]

\[
\text{Write prev-key, save(2,i), save(1,i), save(3,i)}
\]

\[
\text{limit-count = limit-count + 1.}
\]

First-Trip:

\[
\text{For } i = 1 \text{ to trip-agg-code}
\]

\[
\text{If save-sched-dev(1,i) < early or save-sched-dev(1,i) > late}
\]

\[
\text{Write prev-key, save(1,i), 'first-trip', save(2)}
\]

\[
\text{limit-count = limit-count + 1.}
\]

Last-Trip:

\[
\text{Call check-tolerances.}
\]

\[
\text{For } i = 1 \text{ to trip-agg-code}
\]

\[
\text{If save-sched-dev(3,i) < early or save-sched-dev(3,i) > late}
\]

\[
\text{Write prev-key, save(3,i), save(2,1), 'last-trip'}
\]

\[
\text{limit-count = limit-count + 1.}
\]

\[
\text{Write prev-line, prev-direction, limit-count.}
\]

\[
\text{If new-key = high-values}
\]

\[
\text{prev = new.}
\]

\[
\text{Call read-three-trips}
\]

\[
\text{Call first-trip.}
\]
B.1.3.4.2 Form Daily Overload Report

Program Variables:

save(3,trip-agg-code)
TP
time
adj-sched-dev
max-psgrs
max-psgr-stop
excess-psgrs

overload-percent

Main:

Call process-header.
Call initialize.
Call process-data.

Initialize:

Read selection-criteria into overload-percent.
Read TP-trip into new
at end Call error1.
prev = new.
Call read-three-trips.
If new-key = high-values Call error2.
Call first-trip.

Process-Data:

Do While new-key ≠ high-values

* Trip changes handled in read-trip *
Do While new-line = prev-line
and new-direction = prev-direction

Call check-tolerances.
save(1,*) = save(2,*).
save(2,*) = save(3,*).
prev = new.
Call read-trip(3).

Call last-trip.
Read-Three-Trips:

.limit-count = 0.
save = 0.
Call read-trip(1).
Call read-trip(2).
Call read-trip(3).

Read-Trip(i):

Do While new-key = prev-key

prev = new
capacity = seat-table(prev-bus).
load-limit = overload-percent x capacity.
If prev-TP = begin-code and prev-TP = end-code
save(i,prev-TP) = prev By-Name
If prev-max-psgrs > load-limit
save-excess-psgrs = prev-max-psgrs minus load-limit.
Read TF-trip Into new
at end new-key = high-values.

Check-Tolerances:

For i = 1 to trip-agg-code
If save-excess-psgrs(2,i) > 0
Write prev-key, save(2,i), save(1,i), save(3,i)
limit-count = limit-count + 1.

First-Trip:

For i = 1 to trip-agg-code
If save-excess-psgrs(1,i) > load-limit
Write prev-key, save(1), 'first-trip', save(2).
limit-count = limit-count + 1.

Last-Trip:

Call check-tolerances.
For i = 1 to trip-agg-code
If save-sched-dev(3,i) < early or
save-sched-dev(3,i) > late
Write prev-key,save(3,i), save(2,i), 'last-trip'
limit-count = limit-count + 1.
Write prev-line, prev-direction, limit-count.
If new-key = high-values
prev = new.
Call read-three-trips
Call first-trip.
B.2 Common Report Program Contents (reports 2.1 through 2.6)

Input file structure is identical to the output file structure of given summary file:

```
new
key
key1
line
direction
day-int
key2
(time-int)  * either time-int or trip  *
(trip)      * will be present; not both *
data (varies with file)
```

prev: same as above.

calc
data (same as above)

There are two outputs:

out-print: print line
out-week: summarized weekly data, same format as input. It is accumulated when the DCW outputs are being printed; then is later read and printed.

Output data line structure includes same data (except possibly for sample-size).

'data' consists of four types:

1. sums (includes sample sizes)
2. averages or percentages
3. variances
4. extremes

Therefore, additional data of the same type can be incorporated by:

1. adding addl-sums to old-sums
2. Calling 'average' (described later), with the old and addl averages (or percentages) and sample sizes
3. Calling "variance (described later), with the old and addl averages, variances and sample sizes.
4. comparing old and addl extremes
The format of the summary files are given here; the data fields that are averages or percentages are indicated by 'a/p'; those that are variances are indicated by 'v'; those that are extremes are indicated by 'e'; the remaining ones are sums.

Section-15-Aggregate File

Program Variables:

out
key
  line
direction
day-int
time-int
data
  psgrs-boarded
  bus-miles
  psgr-miles
  bus-mins
  psgr-mins
  capacity-miles
  seat-miles
  bus-trips * sample size *

Daily-Run-Aggregate File

Program Variables:

out1
key
  line
direction
day-int
  (time-int)
  (trip)
data
  TP
  a/p sched-run-time
  a/p run-time-dev
  a/p adj-run-time-dev
  v run-time-dev-var
  sample-size

B-37
Daily-Ride-Aggregate Files

Program Variables:

out1
  key
  line
  direction
  day-int
  (time-int)
  (trip)
  data
  TP
  psgrs-on
  psgrs-off
  e
  max-psgr
  e
  max-psgr-stop
  standees
  standee-time(3)
  mins-wi-standees
  sample-size

out2
  key
  line
  direction
  day-int
  (time-int)
  (trip)
  data
  a/p
  psgr-trips
  psgr-miles
  load-factor
  standee-mins
  st-min-per-st
  bus-trips
  * sample size *

Daily-Sched-Dev-Aggregate File

Program Variables:

out
  key
  line
  direction
  day-int
  (time-int)
  data
  TP
  a/F
  hist(22)
  sample-size
Daily-Overload-Aggregate File

Program Variables:

out
  key
  line
  direction
  day-int
  time-int
  data
  TP
  a/p hist(10)
  sample-size

Daily-Layover-Aggregate File

Program Variables:

out
  key
  line
  direction
  day-int
  (time-int)
  (trip)
  data
  a/p sched-layover
  a/p layover-dev
  sample-size

Main:

Call process-header.
Call initialize(summary).
Call process-data(summary).
Call process-week.
Call initialize(week).
Call process-data(week).
Initialize(file):

reset out-print, out-week (to zeros or blanks).
calc = 0.
read file into new.
prev = new.
call print-header(new-key).

Process-data(file):

do while new-key <= high-values
  do while new-key = prev-key
    move prev-data into proper column of out-print.
    prev = new.
    read file into new at end new-key = high-values.
  call print-line.
end do

print-line:

increment print-line-counter.
if print-line-counter > page-size
  call print-header(prev-key).
print out-print.
reset out-print.
if new-key <= high-values
  if new-key1 <= prev-key1
    call print-header(new-key).
prev = new.

print-header(key):

* prints formatted key items and other related data (e.g.,
time point names as well as numbers *)
print key-header.
print-line-counter = 0.
Process-week:

* If there is more than one day-of-week category, also want a weekly aggregate, in same format as for individual DOW 'pages' *
If header-DOW-code = all Return.
Call sort-for-day.
Call initialize(summary).

Do While new-key2 $\neq$ high-values
  Do While new-key2 = prev-key2
    Call update.
    prev = new.
    Read summary into new At end new-key2 = high-values.
  Call write-week.

Write-Week:

  out-week-key = prev-key.
  out-week-day-int = day-agg-code.
  out-week = calc By-Name.
  Write week From out-week.
  calc = 0.
  prev = new.

Sort-For-Day:

* Reorder 'summary' so can add successive records (until time-int/trip changes) to get weekly totals *
Rewind summary.
If TOD-code = all
  Sort by line, direction, trip, TP, day-int.
Else Sort by line, direction, time-int, TP, day-int.

Update:

  For i = 1 to max-data-elements
    If prev-data-element(i) is a sum
      calc-data-element(i) = calc-data-element(i) + prev-data-element(i).
    Else If prev-data-element(i) is an average or percentage
      clsd-size = calc-sample-size(i)
      clsd-avg = calc-avg(i)
      Call average(old-size,old-avg,prev-sample-size(i),
        prev-avg(i),calc-sample-size(i),calc-avg(i))
      new-avg = calc-avg(i).
Else If prev data element(i) is a variance
old-var = calc-var(i)
Call variance(old-size,old-avg,old-var,
prev-sample-size(i),prev-avg(i),prev-var(i)
calc-sample-size(i),new-avg,calc-var(i)).
Else If prev-data-element(i) is an extreme
If prev-data-element(i) more-extreme-than
calc-data-element(i)
calc-data-element(i) = prev-data-element(i).

Average(old-size,old-avg,addl-size,addl-avg,new-size,new-avg):

* Method to update an existing average (old) when
another average (addl) becomes available *

new-size = cld-size + addl-size.
If new-size > 0 Then
x12 = old-avg minus addl-avg
n23 = addl-size / new-size
new-avg = old-avg minus n23 x x12.

Else Call error-handling.

Variance(old-size,old-avg,old-var,addl-size,addl-avg,addl-var,
new-size,new-avg,new-var):

* Method to update an existing variance (old) when
another (addl) becomes available *

new-size = cld-size + addl-size.
If new-size > 0 Then
x13sg = old-avg**2 minus new-avg**2
x12sq = old-avg**2 minus addl-avg**2
v12 = old-var minus addl-var
n23 = addl-size / new-size
new-var = old-var + x13sg minus (n23 x (v12 + x12sq)).

Else Call error-handling.