

Logic Online Inc.

# **RIPPLE-TRAC For Source Code and IMS**



©

Written by  
Daniel B. Boucher Sr,  
z/OS Solutions Architect,  
Logic Online, Inc.

A Separation of Concerns (SoC) Tool

[www.logiconlineinc.com](http://www.logiconlineinc.com)

This document contains proprietary information, protected by copyright. No part of this document may be reproduced or transmitted for any purpose other than the reader's personal use without the permission of Logic Online Inc.

## **WARRANTY**

The information contained in this document is subject to change without notice. Logic Online makes no warranty of any kind with respect to this information. Logic Online shall not be liable for any direct, indirect, incidental, consequential, or other damage alleged in connection with the use of this information.

## **TRADEMARKS**

All trademarks and registered trademarks used in this guide are property of their respective owners.

RIPPLE-TRAC is a registered trademark in the state of New Hampshire

Logic Online Inc.  
1500A Lafayette Rd #170  
Portsmouth, NH 03801  
e-mail: [info@legacyimpactanalysis.com](mailto:info@legacyimpactanalysis.com)  
[www.logiconlineinc.com](http://www.logiconlineinc.com)



[www.logiconlineinc.com](http://www.logiconlineinc.com) is a resource for:



<http://www-03.ibm.com/systems/z/destinationz/>

## Overview of RIPPLE-TRAC for Source and IMS

[Multi-dimensional separation of concerns](#) is an approach to separation of concerns, supporting construction, evolution and integration of software. Its goals are to enable:

- Encapsulation of all kinds of concerns in a software system, simultaneously.
- Overlapping and interacting concerns.
- On-demand modularization.

*Separation of concerns* is a concept that is at the core of software engineering. It refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concern (concept, goal, purpose, etc.). Concerns are the primary motivation for organizing and decomposing software into manageable and comprehensible parts. Many kinds of concerns may be relevant to different developers in different roles, or at different stages of the software lifecycle. Appropriate separation of concerns has been hypothesized to reduce software complexity and improve comprehensibility; promote traceability; facilitate reuse, non-invasive adaptation, customization, and evolution; and simplify component integration.

The term *multi-dimensional separation of concerns* (MDSOC) refers to flexible and incremental separation, modularization, and integration of software artifacts based on any number of concerns. It overcomes limitations of existing mechanisms by permitting clean separation of multiple, potentially overlapping and interacting concerns simultaneously. MDSOC promotes reuse, improves comprehension, reduces the impact of change, eases maintenance and evolution, improves traceability, and opens the door to system refactoring and reengineering.

MDSOC summary:

Involves decomposition of software according to one or more dimensions of concern. A concern is any piece of concern or focus in a program.

[http://en.wikipedia.org/wiki/Separation\\_of\\_concerns](http://en.wikipedia.org/wiki/Separation_of_concerns)

The separation allows:

- To allow people to work on individual pieces of the system in isolation;
- To facilitate reusability;
- To ensure the maintainability of a system;
- To add new features easily;
- To enable everyone to better understand the system;
- To allow support for multi-dimensional separation of concerns.

Remember, a dimension of concern is simply an approach to decomposing, organizing, and structuring software according to concerns of a particular kind. **RIPPLE-TRAC** falls into the realm of multi-dimensional separation of concerns.

***The technology engaged by RIPPLE-TRAC presents the information from the point of view of the concern – the architect only sees information that is related to the concern and has control on what concerns should be brought to the forefront or obscured into the background. The uniqueness of RIPPLE-TRAC is the targeted approach, which we believe has potential to support longer term refactoring of application logic into reusable components and services (SOA). RIPPLE-TRAC can also be used as a support tool for migrating from one platform to another.***

The RIPPLE-TRAC result set helps track and manage numerous cross silo, intricate, legacy component mappings. These mapping support situations where components across applications differ in structure. Manual mappings run the risk of missing critical data in the new/composite system. If missing components are discovered during implementation, the project would be delayed or cancelled.

RIPPLE-TRAC is a batch-driven environment that accepts application libraries as input and provides information on execution flow, physical system component relationships, and dependencies. RIPPLE-TRAC includes the ability to rapidly parse and cross-reference system components across multiple applications into an open repository, produce metrics and reports, and allow analysts to examine and extract information on an ad hoc basis. Results are depicted in a cross-reference list as well as a spreadsheet or any relational model chosen by the analyst.

RIPPLE-TRACs batch analyzer is more conducive to planning activities than an interactive static analysis tool. Planning teams can use metrics and summary-level information to assess the complexity of an application. The more complex a system, the more time it takes to analyze, enhance, improve, or transform. If, for example, a group of programs is highly complex, poorly structured, and utilizes a number of constructs that are hard to decipher, it would increase the time and skills needed to extract business logic from those programs.

## **What RIPPLE-TRAC is not**

RIPPLE-TRAC is not a quick and dirty data migration tool that converts database call structures to embedded SQL. SQL is the standard access mode used to read and update data within a relational database . Such an approach sacrifices comprehensiveness, quality and integrity within the resulting relational database for the sake of time. The result is a database that is poorly designed, limited in its accessibility and flexibility, and detrimental to the business units relying on this data.[1]

We will use an IMS hospital data base as an example of the RIPPLE-TRAC process.

IBM **Information Management System (IMS)** is a joint [hierarchical database](#) and [information management](#) system with extensive [transaction processing](#) capabilities. [wikipedia]

A **hierarchical database model** is a [data model](#) in which the data is organized into a [tree](#)-like structure. The structure allows representing information using parent/child relationships: each parent can have many children, but each child has only one parent (also known as a 1-to-many relationship). All attributes of a specific record are listed under an entity type. [wikipedia]

Our result sets can then be viewed on a PC. Our objective with RIPPLE-TRAC is to create a simplified, modern application for supporting / learning and / gaining control of the following:

## **The DBD statement**

A DBD is a series of macro instructions that describes such things as a database's organization and access method, the segments and fields in a database record, and the relationships between types of segments.

This utility is a macro assembler that generates a DBD control block and stores it in the IMS.DBDLIB library for subsequent use during database processing.

## **The SEGM statement**

The SEGM statement defines a segment type in the database, the position of the segment in the hierarchy, the physical characteristics of the segment, and the relationship of the segment to other segments.

SEGM statements are put in the input deck in hierarchical sequence, and a maximum of 15 hierarchical levels can be defined. The number of database statements allowed depends on the type of database. SEGM statements must immediately follow the data set or AREA statements to which they are related.

## **The FIELD statement**

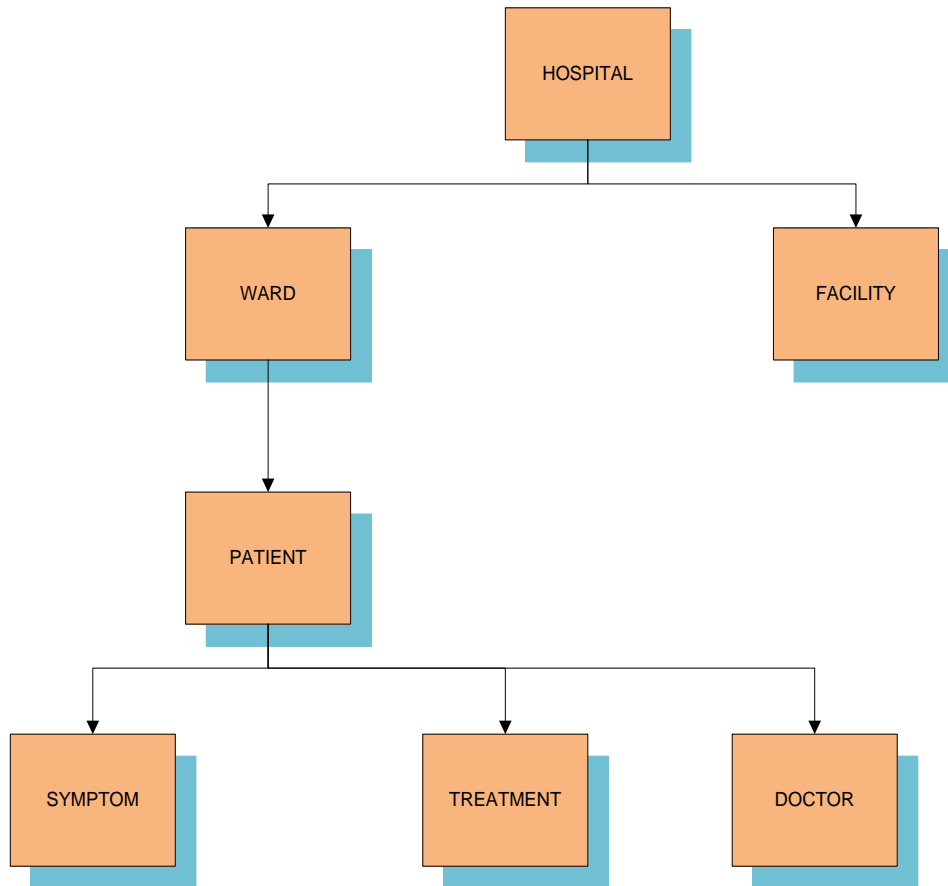
The FIELD statement defines a field within a segment type.

## **IMS PCB details**

An IMS PCB, or program communication block, is defined by the user during PSB generation and describes an application program's interface to, and view of, an IMS database.

The following figure shows the hierarchical structure of the segments in the Hospital database.

Figure 1. Segments of the Hospital database



Each node in the figure represents a segment:

- The HOSPITAL segment is the root segment in the database.
- WARD has a direct descendent segment named PATIENT.
- FACILITY has no descendent segment.
- SYMPTOM, TREATMENT, and DOCTOR are the child segments of the PATIENT.

The tables that follow show the layouts of each segment in the Hospital database.

## HOSPITAL segment

The following table shows the HOSPITAL segment, which has two fields:

- The hospital code (HOSPNAME)
- The hospital name (ADMIN)

HOSPNAME is a unique key field.

Field name	Field length (in bytes)
HOSPNAME	20
ADMIN	20

## WARD segment

The following table shows the WARD segment, which has three fields:

- The ward number (WARDNO)
- The bed available count (BEDAVAIL)
- The ward type (WARDTYPE)

WARDNO is a unique key field.

Field name	Field length (in bytes)
WARDNO	2
BEDAVAIL	3
WARDTYPE	20

## PATIENT segment

The following table shows the PATIENT segment, which has three fields:

- The bed identity (BEDIDENT)
- The patient name (PATNAME)
- The date admitted (DATEADMT)

BEDIDENT is a unique key field.

Field name	Field length (in bytes)
BEDIDENT	4
PATNAME	20

DATEADMT	6
----------	---

## SYMPTON segment

The following table shows the symptom segment, which has two fields:

- The symptom date (SYMPDATE)
- The diagnosis (DIAGNOSE)

Field name	Field length (in bytes)
SYMPDATE	6
DIAGNOSE	20

## TREATMNT segment

The following table shows the TREATMNT segment, which has two fields:

- The day of treatment (TRDATE)
- The type of treatment (TRTYPE)

Field name	Field length (in bytes)
TRDATE	6
TRTYPE	20

## DOCTOR segment

The following table shows the DOCTOR segment, which has two fields:

- The doctor name (DOCTNAME)
- The doctor specialty (SPECIALT)

Field name	Field length (in bytes)
DOCNAME	20
SPECIALT	20

## FACILITY segment

The following table shows the FACILITY segment, which two fields:

- The facility type (FACTYPE)
- The facility availability (FACAVAIL)

Field name	Field length (in bytes)
FACTYPE	20
FACAVAIL	3

## The DBD statement

Once segments are defined, the Hierarchical Structure decided upon, the key and search fields identified, DBA can communicate this design of Database to DL/I

Accordingly a CONTROL BLOCK for Database called DBD is created in a DL/I library, which describes the physical structure of a database to IMS

DBD is created by a process called DBD Generation or DBDGEN, wherein DBA codes a series of Control statements. The process is performed only once for database unless the physical structure of database undergoes a change.

DBDGEN control statements are Assembler Language Macros supplied by IBM and submitted via JCL invoking a catalogued procedure called DBDGEN producing an OBJECT module.

Macros are stored in Maclib and load modules are stored in DBDLIB, which is the DBD itself and ready for loading

The Hospital database DBD control statements will look like:

```
PRINT
NOGEN
DBD          NAME=HOSPDBD, ACCESS=HISAM
DATASET     DD1=PRIME, OVFLW=HOSPFLW, DEVICE=3330
SEGM        NAME=HOSPITAL, PARENT=0, BYTES=60
FIELD       NAME=(HOSPNAME,SEQ,U), BYTES=20, START=1, TYPE=C
SEGM        NAME=WARD, PARENT=HOSPITAL, BYTES=31
FIELD       NAME=(WARDNO,SEQ,U), BYTES=2, START=1, TYPE=C
FIELD       NAME=BEDAVAIL, BYTES=3, START=9, TYPE=C
FIELD       NAME=WARDTYPE, BYTES=20, START=12, TYPE=C
SEGM        NAME=PATIENT, PARENT=WARD, BYTES=70
FIELD       NAME=(BEDIDENT,SEQ,U), BYTES=4, START=61, TYPE=C
FIELD       NAME=PATNAME, BYTES=20, START=1, TYPE=C
FIELD       NAME=DATEADMT, BYTES=6, START=65, TYPE=C
SEGM        NAME=SYMPTOM, PARENT=PATIENT, BYTES=76
FIELD       NAME=(SYMPDATE,SEQ), BYTES=6, START=21, TYPE=C
FIELD       NAME=DIAGNOSE, BYTES=20, START=1, TYPE=C
SEGM        NAME=TREATMNT, PARENT=PATIENT, BYTES=113
FIELD       NAME=(TRDATE,SEQ), BYTES=6, START=21, TYPE=C
FIELD       NAME=TRTYPE, BYTES=20, START=1, TYPE=C
SEGM        NAME=DOCTOR, PARENT=PATIENT, BYTES=80
FIELD       NAME=DOCTNAME, BYTES=20, START=1, TYPE=C
FIELD       NAME=SPECIALT, BYTES=20, START=61, TYPE=C
SEGM        NAME=FACILITY, PARENT=HOSPITAL, BYTES=26
FIELD       NAME=FACTYPE, BYTES=20, START=1, TYPE=C
FIELD       NAME=FACAVAIL, BYTES=3, START=24, TYPE=C
DBDGEN
FINISH
```

## **APPLICATION / LOGICAL DATA- STRUCTURE (PSB)**

Logical Data Structure is a term used to refer to a group of segments that an application program may access. To the application program, the logical data structure is the database

-

The DBA defines the logical data structure by creating a Control Block called a Program Specification Block or PSB.

PSB is created with a process called PSB generation or PSBGEN, very similar to the DBDGEN process; wherein DBA codes a series of control statements

PSBGEN control statements are Assembler Language Macros supplied by IBM and submitted via JCL invoking a catalogue procedure called PSBGEN producing an OBJECT module.

Macros are stored in Maclib and Load modules are stored in PSBLIB, which is the PSB itself and ready for loading

The PSB control statements for a Retrieved program will look as below.

Each PSB consists of one or more control blocks called Program Communication Blocks or PCB's. Each PCB within a PSB defines exactly one Logical Data Structure. All the PCB's within a single PSB are collectively known as an Application Data Structure

It is possible that more than one program may share a single PSB, however, no program can use more than one PSB in a single execution

The Logical Data Structures defines all the segments that an application program may access using that PSB

PRINT NOGEN

PCB TYPE=DB,DBDNAME=HOSPDBD,PROCOPT=AP,KEYLEN=32

\*

SENSEG NAME=HOSPITAL,PARENT=0

SENSEG NAME=WARD,PARENT=HOSPITAL

SENSEG NAME=PATIENT,PARENT=WARD

SENSEG NAME=SYMPTOM,PARENT=PATIENT

SENSEG NAME=TREATMNT,PARENT=PATIENT

SENSEG NAME=DOCTOR,PARENT=PATIENT

SENSEG NAME=FACILITY,PARENT=HOSPITAL

\*

PSBGEN LANG=COBOL,PSBNAME=HOSPPSB

END

# RIPPLE-TRAC Run Time Behavior For IMS

The intent is to find all references to the fields in the hospital data base and map them to a new architecture. This includes all PSB's, PCB's, logical DBD's any code including COPYBOOKS, INCLUDES etc.

The ITEMS OF CONCERN in the 'HOSPDBD' DBDGEN for the hospital data base are used as input to RIPPLE-TRAC:

```
*****IMS SECTION*****I
DLITCBL          DLI INTERFACE          I
PLITDLI          PL1 INTERFACE          I
CBLTDLI          COBOL INTERFACE        I
****HOSPITAL DATA BASE
HOSPDBD          DBD NAME              I
HOSPITAL         .SEG NAME             I
HOSPNAME         ..FIELD NAME         I
ADMIN            ..FIELD NAME         I
WARD             .SEG NAME             I
WARDNO          ..FIELD NAME         I
BEDAVAIL         ..FIELD NAME         I
WARDTYPE        ..FIELD NAME         I
PATIENT         .SEG NAME             I
BEDIDENT        ..FIELD NAME         I
PATNAME         ..FIELD NAME         I
DATEADMT        ..FIELD NAME         I
SYMPTOM         .SEG NAME             I
SYMPDATE        ..FIELD NAME         #
DIAGNOSE        ..FIELD NAME         #
TREATMNT        .SEG NAME             I
TRDATE          ..FIELD NAME         I
TRTYPE          ..FIELD NAME         I
DOCTOR          .SEG NAME             I
DOCTNAME        ..FIELD NAME         I
SPECIALT        ..FIELD NAME         I
FACILITY        .SEG NAME             I
FACTYPE         ..FIELD NAME         I
FACAVAIL        ..FIELD NAME         I
****END HOSPITAL DATA BASE
```

The above fields are gathered manually in this release. (V1.5.01)

RIPPLE-TRAC result set has found HOSPITAL references in 3 different pieces of code.

HOSPCPY	COPYBOOK
HOSPPSB	PSB
LOADHTL	COBOL PROGRAM

## HOSPCPY RESULT SET

REL-SEQ	MODULE	TYPE	EXISTING	BACK-REF	ITEMS LABEL	ITEMS OF CONCERN	TARGET DESCRIPTION
1	HOSPCPY	I	..SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
2	HOSPCPY	I	..FIELD NAME			HOSPNAME	DB2 MODEL A TABLE 6 FIELD 1
5	HOSPCPY	I	..FIELD NAME			ADMIN	DB2 MODEL A TABLE 6 FIELD 2
6	HOSPCPY	*			COPY	TESTCPY	COPY BOOK NOT IN CURRENT COPYLIB
7	HOSPCPY	I	..SEG NAME			WARD	DB2 MODEL A TABLE 7
8	HOSPCPY	I	..SEG NAME			WARD	DB2 MODEL A TABLE 7
8	HOSPCPY	I	..FIELD NAME			WARDNO	DB2 MODEL A TABLE 7 FIELD 1
9	HOSPCPY	I	..SEG NAME			ROOMS	
11	HOSPCPY	I	..FIELD NAME			BEDAVAIL	DB2 MODEL A TABLE 7 FIELD 2
12	HOSPCPY	I	..SEG NAME			WARD	DB2 MODEL A TABLE 7
12	HOSPCPY	I	..FIELD NAME			WARDTYPE	DB2 MODEL A TABLE 7 FIELD 3
14	HOSPCPY	I	..SEG NAME			PATIENT	DB2 MODEL A TABLE 8
15	HOSPCPY	I	..FIELD NAME			PATNAME	DB2 MODEL A TABLE 8 FIELD 2
18	HOSPCPY	I	..FIELD NAME			BEDIDENT	DB2 MODEL A TABLE 8 FIELD 1
19	HOSPCPY	I	..FIELD NAME			DATEADMT	DB2 MODEL A TABLE 8 FIELD 3
25	HOSPCPY	I	..SEG NAME			SYMPTOM	DB2 MODEL A TABLE 9
26	HOSPCPY	#	..FIELD NAME			DIAGNOSE	DB2 MODEL A TABLE 9 FIELD 2
27	HOSPCPY	#	..FIELD NAME			SYMPDATE	DB2 MODEL A TABLE 9 FIELD 1

Column 1 'REL-SEQ' is the relative sequence number of the line of code.

Column 2 'MODULE' is the module name.

Column 3 'TYPE' is the type of component, in this case, 'I' stands for IMS.

**NOTE: REL-SEQ 26 and 27 TYPE '#' indicates a precision issue during conversion.**

Column 4 'EXISTING' is the existing item label name.

Column 5 'BACK-REF' not used in this example.

Column 6 'ITEMS LABEL' is the type of item label.

Column 7 'ITEMS OF CONCERN' is the actual item of concern.

Column 8 'TARGET DESCRIPTION' is where the item of concern is mapped to.

Refer to the above result set.

As you can see, RIPPLE-TRAC found 15 variations in this HOSPCPY copybook.

```
000001      01  HOSPITAL.
000002          03  HOSPNAME          PIC X(20) .
000003          03  HOSP-ADDRESS      PIC X(30) .
000004          03  HOSP-PHONE        PIC X(10) .
000005          03  ADMIN             PIC X(20) .
000006      01  COPY TESTCPY.
000007      01  WARD.
000008          03  WARDNO            PIC XX.
000009          03  TOT-ROOMS         PIC XXX.
000010          03  TOT-BEDS          PIC XXX.
000011          03  BEDAVAIL          PIC XXX.
000012          03  WARDTYPE          PIC X(20) .
000013
000014      01  PATIENT.
000015          03  PATNAME            PIC X(20) .
000016          03  PATADDRESS         PIC X(30) .
000017          03  PAT-PHONE          PIC X(10) .
000017          03  PAT-PHONE          PIC X(10) .
000018          03  BEDIDENT           PIC X(4) .
000019          03  DATEADMT          PIC X(6) .
000020          03  PREV-STAY-FLAG      PIC X.
000021          03  PREV-HOSP          PIC X(20) .
000022          03  PREV-DATE          PIC X(4) .
000023          03  PREV-REASON        PIC X(30) .
000024
000025      01  SYMPTOM.
000026          03  DIAGNOSE            PIC X(20) .
000027          03  SYMPDATE            PIC X(6) .
```

# HOSPPSB RESULT SET

REL-SEQ	MODULE	TYPE	EXISTING	BACK-REF	ITEMS LABEL	ITEMS OF CONCERN	TARGET DESCRIPTION
2	HOSPPSB	I	DBD NAME			HOSPDBD	DB2 MODEL A
4	HOSPPSB	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
5	HOSPPSB	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
5	HOSPPSB	I	.SEG NAME			WARD	DB2 MODEL A TABLE 7
6	HOSPPSB	I	.SEG NAME			WARD	DB2 MODEL A TABLE 7
6	HOSPPSB	I	.SEG NAME			PATIENT	DB2 MODEL A TABLE 8
7	HOSPPSB	I	.SEG NAME			PATIENT	DB2 MODEL A TABLE 8
7	HOSPPSB	I	.SEG NAME			SYMPTOM	DB2 MODEL A TABLE 9
8	HOSPPSB	I	.SEG NAME			PATIENT	DB2 MODEL A TABLE 8
8	HOSPPSB	I	.SEG NAME			TREATMNT	DB2 MODEL A TABLE 1
9	HOSPPSB	I	.SEG NAME			PATIENT	DB2 MODEL A TABLE 8
9	HOSPPSB	I	.SEG NAME			DOCTOR	DB2 MODEL A TABLE 2
10	HOSPPSB	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
10	HOSPPSB	I	.SEG NAME			FACILITY	DB2 MODEL A TABLE 3
12	HOSPPSB	A				LANG=COBOL	

Column 1 'REL-SEQ' is the relative sequence number of the line of code.  
 Column 2 'MODULE' is the module name.  
 Column 3 'TYPE' is the type of component, in this case, 'I' stands for IMS.  
 Column 4 'EXISTING' is the existing item label name.  
 Column 5 'BACK-REF' not used in this example.  
 Column 6 'ITEMS LABEL' is the type of item label.  
 Column 7 'ITEMS OF CONCERN' is the actual item of concern.  
 Column 8 'TARGET DESCRIPTION' is where the item of concern is mapped to.

Refer to the above result set.

As you can see, RIPPLE-TRAC found 9 variations in this HOSPPSB PSB.

```

000001 PRINT NOGEN
000002   PCB TYPE=DB, DBDNAME=HOSPDBD, PROCOPT=AP, KEYLEN=32
000003 *
000004   SENSEG NAME=HOSPITAL, PARENT=0
000005   SENSEG NAME=WARD, PARENT=HOSPITAL
000006   SENSEG NAME=PATIENT, PARENT=WARD
000007   SENSEG NAME=SYMPTOM, PARENT=PATIENT
000008   SENSEG NAME=TREATMNT, PARENT=PATIENT
000009   SENSEG NAME=DOCTOR, PARENT=PATIENT
000010   SENSEG NAME=FACILITY, PARENT=HOSPITAL
000011 *
000012   PSBGEN LANG=COBOL, PSBNAME=HOSPPSB
000013   END
  
```

## LOADHTL RESULT SET

REL-SEQ	MODULE	TYPE	EXISTING	BACK-REF	ITEMS LABEL	ITEMS OF CONCERN	TARGET DESCRIPTION
229	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
230	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
232	LOADHTL	I	..FIELD NAME			HOSPNAME	DB2 MODEL A TABLE 6 FIELD 1
234	LOADHTL	I	..FIELD NAME			HOSPNAME	DB2 MODEL A TABLE 6 FIELD 1
533	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
536	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
701	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
702	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
704	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
705	LOADHTL	I	..FIELD NAME			HOSPNAME	DB2 MODEL A TABLE 6 FIELD 1
705	LOADHTL	I	..FIELD NAME			HOSPNAME	DB2 MODEL A TABLE 6 FIELD 1
709	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
711	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
713	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
714	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
714	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
721	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
722	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
725	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
734	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
735	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6
737	LOADHTL	I	.SEG NAME			HOSPITAL	DB2 MODEL A TABLE 6

Column 1 'REL-SEQ' is the relative sequence number of the line of code.  
 Column 2 'MODULE' is the module name.  
 Column 3 'TYPE' is the type of component, in this case, 'I' stands for IMS.  
 Column 4 'EXISTING' is the existing item label name.  
 Column 5 'BACK-REF' not used in this example.  
 Column 6 'ITEMS LABEL' is the type of item label.  
 Column 7 'ITEMS OF CONCERN' is the actual item of concern.  
 Column 8 'TARGET DESCRIPTION' is where the item of concern is mapped to.

Refer to the above result set.

As you can see, RIPPLE-TRAC found 22 locations in the LOADHTL COBOL program.

```

000229      01  QU-T1-HOSPITAL-SSA.
000230          05                      PIC X(8)    VALUE 'HOSPITAL'.
000232          05                      PIC X(8)    VALUE 'HOSPNAME'.
000234          05  QU-T1-HOSPNAME      PIC X(3) .

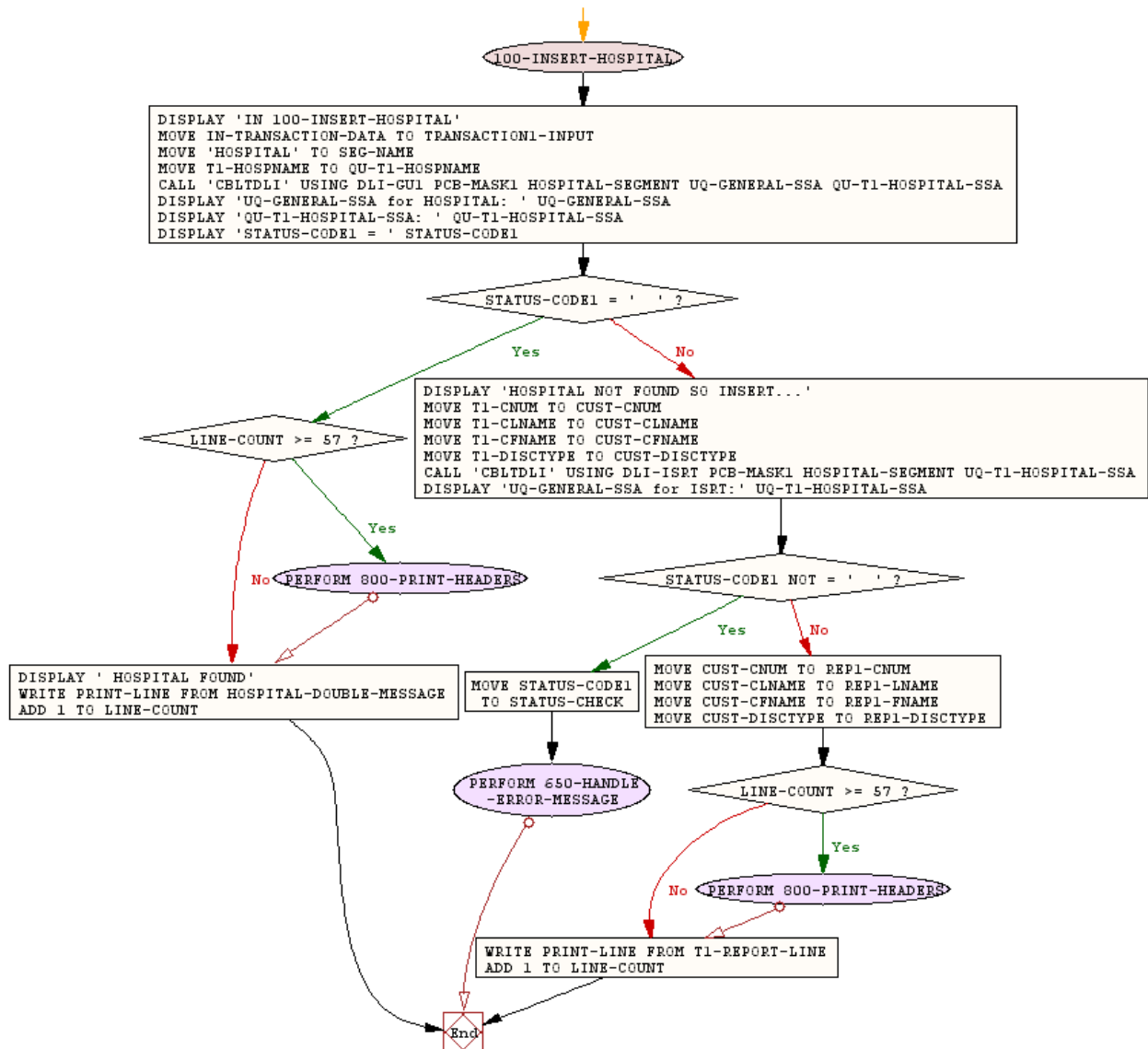
000533      01  HOSPITAL-DOUBLE-MESSAGE.
000534          05                      PIC X(10)   VALUE SPACES.
000535          05                      PIC X(37)
000536          VALUE '***** HOSPITAL ALREADY EXISTS *****'.

000701      100-INSERT-HOSPITAL.
000702          DISPLAY 'IN 100-INSERT-HOSPITAL'.
000704          MOVE 'HOSPITAL' TO SEG-NAME.
000705          MOVE T1-HOSPNAME TO QU-T1-HOSPNAME.
000709                      HOSPITAL-SEGMENT
000711                      QU-T1-HOSPITAL-SSA.
000713          DISPLAY 'UQ-GENERAL-SSA for HOSPITAL: ' UQ-GENERAL-SSA.
000714          DISPLAY 'QU-T1-HOSPITAL-SSA: ' QU-T1-HOSPITAL-SSA.
000721          DISPLAY ' HOSPITAL FOUND'
000722          WRITE PRINT-LINE FROM HOSPITAL-DOUBLE-MESSAGE
000725          DISPLAY 'HOSPITAL NOT FOUND SO INSERT... '
000734                      HOSPITAL-SEGMENT
000735                      UQ-T1-HOSPITAL-SSA
000737          DISPLAY 'UQ-GENERAL-SSA for ISRT:' UQ-T1-HOSPITAL-SSA

```

By flowcharting this routine we can get a better idea of the business logic.

See flowchart of paragraph '100-INSERT-HOSPITAL' below:



Another example, FILTERING LOGIC:

We want to see all the precision issues in the result set with corresponding target system.

As indicated above, **TYPE '#' indicates a precision issue during conversion.**

REL-SEQ	MODULE	TYPE	EXISTING	BACK-REF	ITEM LABEL	ITEMS OF CONCERN	TARGET DESCRIPTION
26	HOSPCPY	#	..FIELD NAME			DIAGNOSE	DB2 MODEL A TABLE 9 FIELD 2
27	HOSPCPY	#	..FIELD NAME			SYMPDATE	DB2 MODEL A TABLE 9 FIELD 1
22	HOSPDBD	#	..FIELD NAME			SYMPDATE	DB2 MODEL A TABLE 9 FIELD 1
23	HOSPDBD	#	..FIELD NAME			DIAGNOSE	DB2 MODEL A TABLE 9 FIELD 2

Column 1 'REL-SEQ' is the relative sequence number of the line of code.

Column 2 'MODULE' is the module name.

Column 3 'TYPE' is the type of component, in this case, 'I' stands for IMS.

**NOTE: REL-SEQ 26, 27, 22, 23 TYPE '#' indicates a precision issue during conversion.**

Column 4 'EXISTING' is the existing item label name.

Column 5 'BACK-REF' not used in this example.

Column 6 'ITEMS LABEL' is the type of item label.

Column 7 'ITEMS OF CONCERN' is the actual item of concern.

Column 8 'TARGET DESCRIPTION' is where the item of concern is mapped to.

As you can see, 4 precision issues exist that will require additional analysis either by creating an intermediate file before moving to target system or allowing truncation.

## Another example C++ IMS:

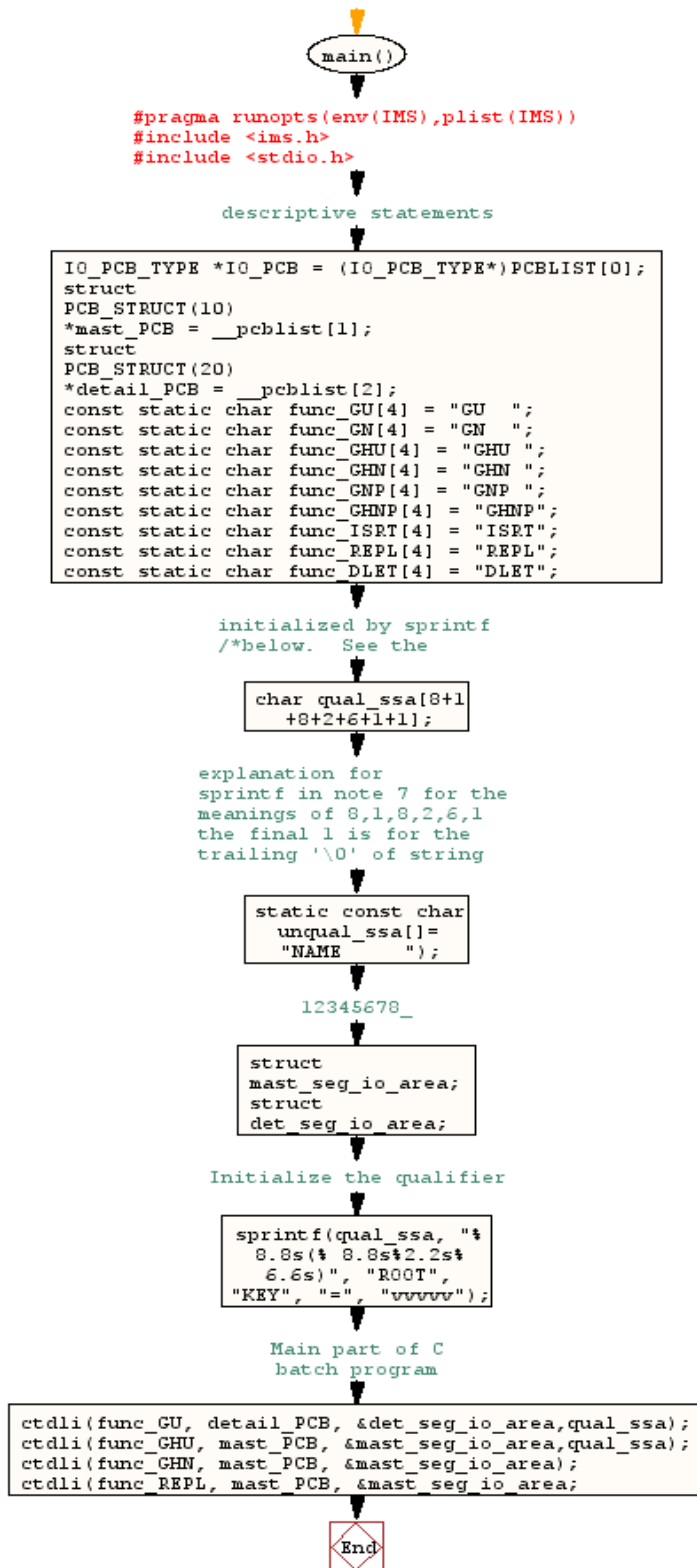
REL-SEQ	MODULE	TYPE	EXISTING	BACK-REF	ITEM LABEL	ITEM OF CONCERN	DESCRIPTION
2	C	*			INCLUDE	<ims.h>	COPY BOOK NOT IN CURRENT COPYLIB
3	C	*			INCLUDE	<stdio.h>	COPY BOOK NOT IN CURRENT COPYLIB
9	C	+				mast_PCB	PROGRAM CONTROL BLOCK
9	C	+				struct	C++ STRUCT STATEMENT
10	C	+				struct	C++ STRUCT STATEMENT
13	C	+				GHU	GET HOLD UNIQUE
13	C	+				GHU	GET HOLD UNIQUE
15	C	I	FUNCTION			GNP	
15	C	I	FUNCTION			GNP	
29	C	+				struct	C++ STRUCT STATEMENT
33	C	+				mast_seg_io_area	MASTER SEGMENT I/O AREA
35	C	+				struct	C++ STRUCT STATEMENT
53	C	+				GHU	GET HOLD UNIQUE
53	C	+				mast_PCB	PROGRAM CONTROL BLOCK
54	C	+				mast_seg_io_area	MASTER SEGMENT I/O AREA
56	C	+				mast_PCB	PROGRAM CONTROL BLOCK
57	C	+				mast_seg_io_area	MASTER SEGMENT I/O AREA
59	C	+				mast_PCB	PROGRAM CONTROL BLOCK
60	C	+				mast_seg_io_area	MASTER SEGMENT I/O AREA

Column 1 'REL-SEQ' is the relative sequence number of the line of code.  
 Column 2 'MODULE' is the module name.  
 Column 3 'TYPE' is the type of component, in this case, 'I' stands for IMS.  
 Column 4 'EXISTING' is the existing item label name.  
 Column 5 'BACK-REF' not used in this example.  
 Column 6 'ITEMS LABEL' is the type of item label.  
 Column 7 'ITEMS OF CONCERN' is the actual item of concern.  
 Column 8 'DESCRIPTION' is the description of the item of concern.

NOTE: This code is not part of the conversion, hence, no target DB2 tables.

```
000002 #include <ims.h>
000003 #include <stdio.h>
000009 struct {PCB_STRUCT(10)} *mast_PCB = __pcblist[1];
000010 struct {PCB_STRUCT(20)} *detail_PCB = __pcblist[2];
000013 const static char func_GHU[4] = "GHU ";
000015 const static char func_GNP[4] = "GNP ";
000029 struct {
000033     } mast_seg_io_area;
000035 struct {
000053     ctdli(func_GHU, mast_PCB,
000054         &mast_seg_io_area, qual_ssa);
000056     ctdli(func_GHN, mast_PCB,
000057         &mast_seg_io_area);
000059     ctdli(func_REPL, mast_PCB,
000060         &mast_seg_io_area);
```

By flowcharting this routine we can get a better idea of the business logic.



## Another example PL/1 IMS:

REL-SEQ	MODULE	TYPE	EXISTING	BACK-REF	ITEM LABEL	ITEM OF CONCERN	DESCRIPTION
17	PL1IMS	+				GHU	GET HOLD UNIQUE
17	PL1IMS	+				GHU	GET HOLD UNIQUE
18	PL1IMS	P				DCL FUNC_GHN	
19	PL1IMS	I	FUNCTION			GNP	
19	PL1IMS	I	FUNCTION			GNP	
19	PL1IMS	P				DCL FUNC_GNP	
20	PL1IMS	P				DCL FUNC_GHN	
20	PL1IMS	P				DCL FUNC_GHNP	
21	PL1IMS	P				DCL FUNC_ISRT	
22	PL1IMS	P				DCL FUNC_REPL	
23	PL1IMS	P				DCL FUNC_DLET	
26	PL1IMS	P				CHAR(8) INIT('ROOT ')	
27	PL1IMS	P				CHAR(1) INIT('(')	
28	PL1IMS	P				CHAR(8) INIT('KEY ')	
29	PL1IMS	P				CHAR(2) INIT('=')	
30	PL1IMS	P				CHAR(6) INIT('vvvvv')	
33	PL1IMS	P				CHAR(8) INIT('NAME ')	
76	PL1IMS	I	PL1 INTERFACE			PLITDLI	
79	PL1IMS				DYNAMIC CALL	PLITDLI	
79	PL1IMS	I	PL1 INTERFACE			PLITDLI	
79	PL1IMS	+				GHU	GET HOLD UNIQUE
82	PL1IMS				DYNAMIC CALL	PLITDLI	
82	PL1IMS	I	PL1 INTERFACE			PLITDLI	
85	PL1IMS				DYNAMIC CALL	PLITDLI	
85	PL1IMS	I	PL1 INTERFACE			PLITDLI	
96	PL1IMS	+				GHU	GET HOLD UNIQUE

Column 1 'REL-SEQ' is the relative sequence number of the line of code.  
 Column 2 'MODULE' is the module name.  
 Column 3 'TYPE' is the type of component, in this case, 'I' stands for IMS.  
 Column 4 'EXISTING' is the existing item label name.  
 Column 5 'BACK-REF' not used in this example.  
 Column 6 'ITEMS LABEL' is the type of item label.  
 Column 7 'ITEMS OF CONCERN' is the actual item of concern.  
 Column 8 'DESCRIPTION' is the description of the item of concern.

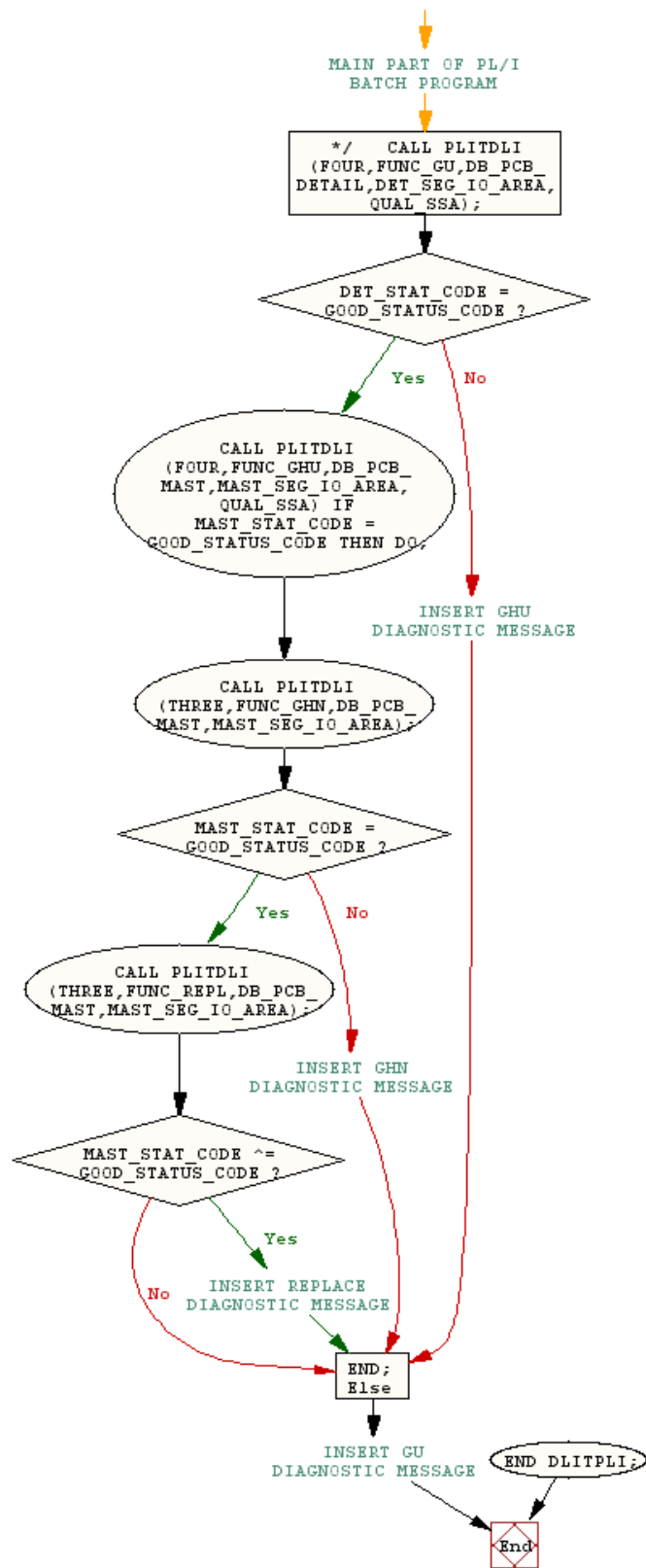
NOTE: This code is not part of the conversion, hence, no target DB2 tables.

```

000017 DCL FUNC_GHU CHAR(4) INIT('GHU ');
000018 DCL FUNC_GHN CHAR(4) INIT('GHN ');
000019 DCL FUNC_GNP CHAR(4) INIT('GNP ');
000020 DCL FUNC_GHNP CHAR(4) INIT('GHNP');
000021 DCL FUNC_ISRT CHAR(4) INIT('ISRT');
000022 DCL FUNC_REPL CHAR(4) INIT('REPL');
000023 DCL FUNC_DLET CHAR(4) INIT('DLET');
000026 2 SEG_NAME CHAR(8) INIT('ROOT '),
000027 2 SEG_QUAL CHAR(1) INIT('('),
000028 2 SEG_KEY_NAME CHAR(8) INIT('KEY '),
000029 2 SEG_OPR CHAR(2) INIT(' ='),
000030 2 SEG_KEY_VALUE CHAR(6) INIT('vvvvv'),
000033 2 SEG_NAME_U CHAR(8) INIT('NAME '),
000076 CALL PLITDLI (FOUR,FUNC_GU,DB_PCB_DETAIL,DET_SEG_IO_AREA, QUAL_SSA);
000079 CALL PLITDLI (FOUR,FUNC_GHU,DB_PCB_MAST,MAST_SEG_IO_AREA,QUAL_SSA)
000082 CALL PLITDLI (THREE,FUNC_GHN,DB_PCB_MAST,MAST_SEG_IO_AREA);
000085 CALL PLITDLI (THREE,FUNC_REPL,DB_PCB_MAST,MAST_SEG_IO_AREA);
000096 /* INSERT GHU DIAGNOSTIC MESSAGE */

```

By flowcharting this routine we can get a better idea of the business logic.



## Another example Assembler IMS:

REL-SEQ	MODULE	TYPE	EXISTING	BACK-REF	ITEM LABEL	ITEM OF CONCERN	DESCRIPTION
23	ASSMIMS				DYNAMIC CALL	ASMTDLI	
27	ASSMIMS				DYNAMIC CALL	ASMTDLI	
27	ASSMIMS	+				GHU	GET HOLD UNIQUE
30	ASSMIMS				DYNAMIC CALL	ASMTDLI	
33	ASSMIMS				DYNAMIC CALL	ASMTDLI	
43	ASSMIMS	+				GHU	GET HOLD UNIQUE
43	ASSMIMS	+				GHU	GET HOLD UNIQUE

Column 1 'REL-SEQ' is the relative sequence number of the line of code.  
 Column 2 'MODULE' is the module name.  
 Column 3 'TYPE' is the type of component, in this case, 'I' stands for IMS.  
 Column 4 'EXISTING' is the existing item label name.  
 Column 5 'BACK-REF' not used in this example.  
 Column 6 'ITEMS LABEL' is the type of item label.  
 Column 7 'ITEMS OF CONCERN' is the actual item of concern.  
 Column 8 'DESCRIPTION' is the description of the item of concern.

NOTE: This code is not part of the conversion, hence, no target DB2 tables.

```

000001 PGMSTART CSECT
000002 ES
000003 *           EQUATE REGISTERS
000004 *   USEAGE OF REGISTERS
000005 R1     EQU   1           ORIGINAL PCBLIST ADDRESS
000006 R2     EQU   2           PCBLIST ADDRESS1
000007 R5     EQU   5           PCB ADDRESSSS
000008 R12    EQU  12           BASE ADDRESS
000009 R13    EQU  13           SAVE AREA ADDRESS
000010 R14    EQU  14
000011 R15    EQU  15
000012 *
000013       USING PGMSTART,R12  BASE REGISTER ESTABLISHED
000014       SAVE  (14,12)       SAVE REGISTERS
000015       LR   12,15         LOAD REGISTERS
000016       ST   R13,SAVEAREA+4  SAVE AREA CHAINING
000017       LA   R13,SAVEAREA    NEW SAVE AREA
000018       USING PCBLIST,R2     MAP INPUT PARAMETER LIST
000019       USING PCBNAME,R5     MAP DB PCB
000020       LR   R2,R1          SAVE INPUT PCB LIST IN REG 2
000021       L    R5,PCBDETA     LOAD DETAIL PCB ADDRESS
000022       LA   R5,0(R5)      REMOVE HIGH ORDER END OF LIST FLAG
000023       CALL ASMTDLI,(GU,(R5),DETSEGIO,SSANAME),VL
000024 *
000025 *
000026       L    R5,PCBMSTA     LOAD MASTER PCB ADDRESS
000027       CALL ASMTDLI,(GHU,(R5),MSTSEGIO,SSAU),VL
000028 *
000029 *
000030       CALL ASMTDLI,(GHN,(R5),MSTSEGIO),VL
000031 *
000032 *

```

```

000033          CALL  ASMTDLI , (REPL, (R5) ,MSTSEGIO) ,VL
000034 *
000035 *
000036          L      R13,4(R13)          RESTORE SAVE AREA
000037          RETURN (14,12)          RETURN BACK
000038
000039 *
000040 *          FUNCTION CODES USED
000041 *
000042 GU          DC      CL4'GU'
000043 GHU          DC      CL4'GHU'
000044 GHN          DC      CL4'GHN'
000045 REPL        DC      CL4'REPL'
000046
000047 *
000048 *          SSAS
000049 *
000050 SSANAME      DS      0C
000051              DC      CL8'ROOTDET'
000052              DC      CL1'('
000053              DC      CL8'KEYDET'
000054 9
000055              DC      CL2' ='
000056 NAME        DC      CL5'  '
000057              DC      C')'
000058 *SSAU        DC      CL9'ROOTMST'*
000059 MSTSEGIO      DC      CL100' '
000060 DETSEGIO      DC      CL100' '
000061 SAVEAREA     DC      18F'0'
000062 *
000063 10
000064 PCBLIST       DSECT
000065 PCBIO         DS      A          ADDRESS OF I/O PCB
000066 PCBMSTA       DS      A          ADDRESS OF MASTER PCB
000067 PCBDETA       DS      A          ADDRESS OF DETAIL PCB
000068 11
000069 *
000070 PCBNAME       DSECT
000071 DBPCBDBD      DS      CL8          DBD NAME
000072 DBPCBLEV      DS      CL2          LEVEL FEEDBACK
000073 DBPCBSTC      DS      CL2          STATUS CODES
000074 DBPCBPRO      DS      CL4          PROC OPTIONS
000075 DBPCBRVS      DS      F          RESERVED
000076 DBPCBSFD      DS      CL8          SEGMENT NAME FEEDBACK
000077 DBPCBMKL      DS      F          LENGTH OF KEY FEEDBACK
000078 DBPCBNSS      DS      F          NUMBER OF SENSITIVE SEGMENTS IN PCB
000079 DBPCBKFD      DS      C          KEY FEEDBACK AREA
000080              END      PGMSTART

```

## **IN SUMMARY**

Software development has seen individual practitioner activities, such as functional and performance testing, become automated. And today's build engineers use timesaving build scripts and tooling. *But more needs to be done to automate the process and steps between roles to improve organizational efficiencies, save money and speed time to market (e.g., RIPPLE-TRAC).*

The **RIPPLE-TRAC** solution can help your team test software more thoroughly and more quickly. Manually analyzing your software applications can result in late releases or inconsistent results. Ad hoc processes for managing your analysis efforts can't ensure the kind of quality you need. By automating your more labor-intensive analysis tasks with **RIPPLE-TRAC**, you can avoid introducing errors into your analysis. The potential payoffs include higher quality and faster time to market—for less money.

Rework is much more costly than building the right solution from the start. To avoid wasteful spending, **RIPPLE-TRAC** is an effective process for defining, capturing, prioritizing and managing modifications to your requirements. **RIPPLE-TRAC** will ensure that IT focuses on the required modifications, so you can deliver what the marketplace is demanding—faster.

## **REFERENCES**

1. LEGACY SYSTEMS Transformation Strategies by William M. Ulrich  
ISBN 0-13-044927-X

A must read for anyone involved in this realm.