

AD-A252 455



WRDC-TR-90-8007
Volume V
Part 28



1

INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume V - Common Data Model Subsystem
Part 28 - Data Aggregators Development Specification

M. Apicella, S. Singh

Control Data Corporation
Integration Technology Services
2970 Presidential Drive
Fairborn, OH 45324-6209



September 1990

Final Report for Period 1 April 1987 - 31 December 1990

Approved for Public Release; Distribution is Unlimited

92-14469



MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

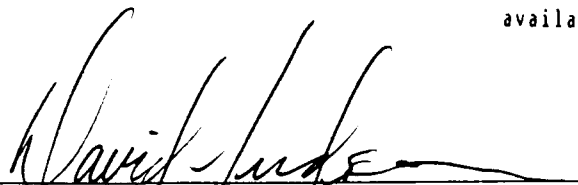
92 6 02 025

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, regardless whether or not the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data. It should not, therefore, be construed or implied by any person, persons, or organization that the Government is licensing or conveying any rights or permission to manufacture, use, or market any patented invention that may in any way be related thereto.

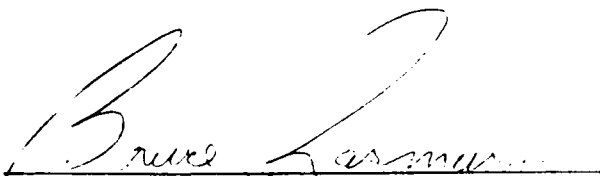
This technical report has been reviewed and is approved for publication.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations


DAVID L. JUDSON, Project Manager
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

FOR THE COMMANDER:


BRUCE A. RASMUSSEN, Chief
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WRDC/MTI, Wright-Patterson Air Force Base, OH 45433-6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release, Distribution is Unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) DS 620341320			5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR- 90-8007 Vol. V, Part 28		
6a. NAME OF PERFORMING ORGANIZATION Control Data Corporation; Integration Technology Services		6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION WRDC/MTI		
6c. ADDRESS (City, State, and ZIP Code) 2970 Presidential Drive Fairborn, OH 45324-6209			7b. ADDRESS (City, State, and ZIP Code) WPAFB, OH 45433-6533		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Wright Research and Development Center, Air Force Systems Command, USAF		8b. OFFICE SYMBOL (if applicable) WRDC/MTI	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM. F33600-87-C-0464		
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB, Ohio 45433-6533			10. SOURCE OF FUNDING NOS.		
11. TITLE (Include Security Classification) See block 19 on			PROGRAM ELEMENT NO. 78011F	PROJECT NO. 595600	TASK NO. 0 F95600
			WORK UNIT NO. 20950607		
12. PERSONAL AUTHOR(S) Control Data Corporation: Apicella, M. L., Singh, S.					
13a. TYPE OF REPORT Final Report		13b. TIME COVERED 4 / 1 / 87 - 12 / 30 / 90	14. DATE OF REPORT (Yr., Mo., Day) 1990 September 30		15. PAGE COUNT 47
16. SUPPLEMENTARY NOTATION WRDC/MTI Project Priority 6203					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify block no.)		
FIELD	GROUP	SUB GR.			
1308	0905				
19. ABSTRACT (Continue on reverse if necessary and identify block number)					
This Development Specification (DS) describes the functions, performance, environment, interfaces, test, qualification, and design requirements for the Data Aggregator computer programs. These programs combine the various sub results from distributed data requests within the Common Data Model subsystem. These programs use the operators JOIN, UNION, and DIFFERENCE. BLOCK 11: INTEGRATED INFORMATION SUPPORT SYSTEM Vol V - Common Data Model Subsystem Part 28 - Data Aggregators Development Specification					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT			ABSTRACT SECURITY CLASSIFICATION		
UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS			Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson			22b. TELEPHONE NO. (Include Area Code) (513) 255-7371		22c. OFFICE SYMBOL WRDC/MTI

FOREWORD

This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Branch Chief, Integration Technology Division, Manufacturing Technology Directorate, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W. A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. Jimmy P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

<u>SUBCONTRACTOR</u>	<u>ROLE</u>
Control Data Corporation	Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS.
D. Appleton Company	Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology.
ONTEK	Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use.
Simpact Corporation	Responsible for Communication development.
Structural Dynamics Research Corporation	Responsible for User Interfaces, Virtual Terminal Interface, and Network Transaction Manager design, development, implementation, and support.
Arizona State University	Responsible for test bed operations and support.

TABLE OF CONTENTS

		<u>Page</u>
SECTION 1.0	SCOPE	1-1
1.1	Identification	1-1
1.2	Functional Summary	1-3
SECTION 2.0	DOCUMENTS	2-1
2.1	Applicable Documents	2-1
2.2	Terms and Abbreviations	2-2
SECTION 3.0	REQUIREMENTS	3-1
3.1	Computer Program Definition	3-1
3.1.1	System Capacities	3-1
3.1.2	Interface Requirements	3-1
3.2	Detailed Functional Requirements	3-1
3.2.1	Function AGG1: Aggregator Inputs ..	3-1
3.2.2	Function AGG2: Perform Union	3-3
3.2.3	Function AGG3: Perform Join	3-5
3.2.4	Function AGG4: Perform Outer Join .	3-17
3.2.5	Function AGG5: Perform NOT-IN-SET Selection	3-25
3.3	Special Requirements	3-32
3.4	Human Performance	3-32
3.5	Database Requirements	3-32
3.6	Adaptation Requirements	3-33
SECTION 4.0	QUALITY ASSURANCE PROVISIONS	4-1
SECTION 5.0	PREPARATION FOR DELIVERY	5-1

LIST OF ILLUSTRATIONS

<u>Figure</u>	<u>Title</u>	<u>Page</u>
1-1	A0 of the CDMP Configuration Items	1-2



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

SECTION 1

SCOPE

1.1 Identification

This specification establishes the performance, development, test and qualification requirements of a collection of computer programs identified as Configuration Item (CI) "Aggregator".

This CI constitutes one of the major subsystems of the "Common Data Model Processor" (CDMP) which is described in the System Design Specification (SDS) for the ICAM Integrated Support System (IISS). The CDMP scope is based on a logical concept of subsystem modules that interface with other external systems of the IISS. The CDMP has been decomposed into three configuration items: the Precompiler, the Distributed Request Supervisor (DRS), and the Aggregator. The scope of the CDMP and its configuration items is described in Figure 1-1 and the following narrative.

Common Data Model Processor (CDMP)

Input to the CDMP consists of user transactions in the form of neutral data manipulation language (NDML) commands embedded in COBOL or FORTRAN host programs. NDML commands phrased as stand-alone requests may be supported in future enhancements.

The Precompiler CI parses the application program source code, identifying NDML commands. It applies external-schema-to-conceptual-schema and conceptual-schema-to-internal-schema transforms on the NDML command, thereby decomposing the NDML command into internal schema single database requests. These single database requests are each transformed into generic DML commands. Programs are generated from the generic DML commands which can access the specific databases to retrieve the data required to evaluate the NDML command. These programs, referred to as Request Processors (RP), are stored at the appropriate host machines. The NDML commands in the application source program are replaced by function calls which, when executed, will activate the run-time request evaluation processes associated with the particular NDML command.

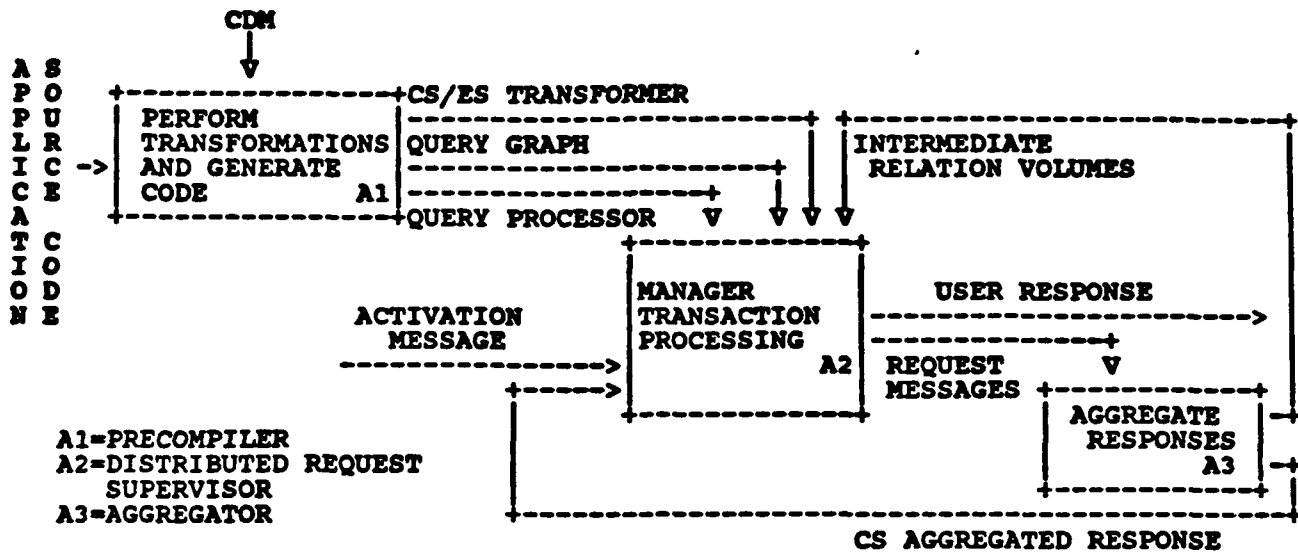


Figure 1-1. A0 of the CDMP Configuration Items

The Precompiler also generates a CS/ES Transformer program which will take the final results of the request, stored in a file as a table with external schema structure, and convert the data values into their external schema form.

Finally, the Precompiler generates a Join Request Graph and Result Field Table, which are used by the Distributed Request Supervisor (DRS) during the run-time evaluation of the request.

The DRS CI is responsible for coordination of the run-time activity associated with the evaluation of an NDML command. It is activated by the application program, which sends it the names and locations of the RPs to activate, along with run-time parameters which are to be sent to the RPs. The DRS activates the RPs, sending them the run-time parameters. The results of the RPs are stored as files, in the form of conceptual schema relations, on the host which executed the RP. Using the Join Request transmission cost information and data about intermediate results, the DRS determines the optimal strategy for combining the intermediate results of the NDML command. It issues the appropriate file transfer requests, activates aggregators to perform join, outer join, union, and not-in-set operations, and activates the appropriate CS/ES Transformer program to transform the final results. Finally, the DRS notifies the application program that the request is completed, and sends it the name of the file which contains the results of the request.

The Aggregator CI is activated by the DRS or by a subroutine call from a user application program. An instance of

the Aggregator is executed for each join, union, not-in-set, or outer join performed. It is passed information describing the operation to be performed and the file names containing the operands of the operation. The DRS or user's application program ensures that these files already exist on the host which is executing the particular Aggregator program. The Aggregator performs the requested operation, storing the results in a file whose name was specified by the DRS or user's application program and is located on the host executing the Aggregator.

The CDMP provides the application programmer with important capabilities to:

1. Request database accesses in a non-procedural data manipulation language (the NDML) that is independent of the data manipulation language (DML) of any particular Data Base Management System (DBMS)
2. Request database access using a DML that specifies accesses to a set of related records rather than to individual records, i.e. using a relational DML
3. Request access to data that are distributed across multiple databases with a single DML command, without knowledge of data locations or distribution details

Information about external schemas, the conceptual schema, and internal schemas (including data locations) are provided by CDMP access to the Common Data Model (CDM) database. The CDM is a relational database of metadata pertaining to IISS. It is described by the CDM1 information model using IDEF1.

Please refer to the Software Availability Bulletin, Volume III, Part 16, CI# SAB620326000, for current IISS software and documentation availability.

1.2 Functional Summary

The overall objective of this CI is to perform relation join, union, outer join and not-in-set operations upon intermediate re- sults of a multi-database transaction. It, along with the DRS, Request Processors, and local DBMS modules, performs the run-time evaluation of commands presented to them by application processes.

There are two Aggregator programs residing on each site. They are identical except that one is invokable via an NTM message while the other is callable as a subroutine. The DRS determines which to use for each aggregation. This development specification describes the generic Aggregator CI, which is to be implemented as multiple Aggregator programs. The DRS or application program causes an Aggregator program to be executed for each join, union, outer join or not-in-set operation which is required to evaluate a particular NDML request. It is the DRS's responsibility to ensure that both operands to the operation already reside in files on the site where the operation is to be performed. When an Aggregator is activated,

it is passed the parameters needed. These parameters include the operand and the result file name. A logical channel ID is also a part of the activation message. This is a communications channel created between the Aggregator and the DRS. The reply message is sent by the Aggregator over this channel when the operation is completed.

The Aggregator sorts the files containing the operands, creates the file which will contain the resultant table, reads the operand files, performs the join, union, outer join or not-in-set operation and stores the results in the resultant relation file. Finally, it deletes the operand files, sends a completion message to the DRS or user application program which created it, and terminates.

Major functions to be described in this document for this CI are:

- o Function AGG1 Aggregator Inputs
- o Function AGG2 Perform Union
- o Function AGG3 Perform Join
- o Function AGG4 Perform Outer Join
- o Function AGG5 Perform Not-In-Set Selection

SECTION 2

DOCUMENTS

2.1 Applicable Documents

Following is a list of applicable documents relating to this Computer Program Development Specification for the Common Data Model Processor (CDMP) Aggregator.

Related ICAM Documents included:

UM620341001	<u>CDM Administrator's Manual</u>
TBM620341000	<u>CDM1, An IDEF1 Model of the Common Data Model</u>
UM620341100	<u>Neutral Data Definition Language (NDDL) User's Guide</u>
PRM620341200	<u>Embedded NDML Programmer's Reference Manual</u>
UM620341002	<u>ICAM Definition Method for Data Modeling (IDEF1-Extended)</u>
DS620341200	<u>Development Specification for the IISS NDML Precompiler Configuration Item</u>
DS620341310	<u>Development Specification for the IISS Distributed Request Supervisor Configuration Item</u>

Other references include:

Bernstein, P. A. et. al., "Request Processing in a System for Distributed Databases (SDD-1)," ACM Transactions on Database Systems, Vol 6, No 4, December 1981.

Blasgen, M. W., Eswaran, K. P., "On the Evaluation of Queues in A Relational Data Base System," IBM Research Report RJ1745, IBM Research Laboratory, San Jose, CA, April 1976.

Chamberlin, D. D., "Relational Data Base Management Systems," Computing Surveys, Vol 8, No 1, May 1976.

Daniels, D., et. al., "An Introduction to Distributed Request Compilation in R*," IBM Research Report RJ3497, IBM Research Laboratory, San Jose, CA, June 1982.

Dejean, J.P., Test Bed System Development Specifications, General Electric Co., November 9, 1982.

Epstein, R., Stonebraker, M., and Wong, E., "Distributed Request Processing in a Relational Database System," Proceedings of the ACM SIGMOD International Conference, Austin, TX, June, 1978.

Lindsay, B. G., et. al., "Notes on Distributed Databases," IBM Research Report RJ2571, IBM Research Laboratory, San Jose, CA, July 14, 1979.

Williams, R., et. al., "R*: An Overview of the Architecture," IBM Research Report RJ3325, IBM Research Laboratory, San Jose, CA, December 12, 1981.

2.2 Terms and Abbreviations

The following acronyms are used in this document:

APL Attribute Pair List
AUC Attribute Use Class
CDMP Common Data Model Processor
CI Configuration Item
CS Conceptual Schema
DML Data Manipulation Language
DRS Distributed Request Supervisor
(previously SS - Stager/Scheduler)
ES External Schema
ICAM Integrated Computer Aided Manufacturing
IISS Integrated Information Support System
IS Internal Schema
NDML Neutral Data Manipulation Language
NIS Not In Set
NTM Network Transaction Manager
RP Request Processor
RFT Result Field Table
SDS System Design Specification

SECTION 3

REQUIREMENTS

3.1 Computer Program Definition

3.1.1 System Capacities

The capacity of the Aggregator is limited by the capacity of the file systems where the operands and resultant relations are stored and by the sorting capabilities of the computer. It is the responsibility of each site's system administrator to ensure that file system and memory allocation capacities are adequate.

3.1.2 Interface Requirements

3.1.2.1 Interface Blocks

This CI is the mechanism that aggregates results of logically related intermediate responses to a distributed database request. Its interfaces include input in the form of two request formats and output in the form of a table (relation) and status responses.

3.1.2.2 Detailed Interface Definition

The specific interface relationships of this CI to other CIs and modules are described in detail for appropriate functions in Section 3.2.

3.2 Detailed Functional Requirements

The following subsections document each of the Aggregator's major functions identified in Section 1.2.

3.2.1 Function AGG1: Aggregator Inputs

This function establishes all inputs required for each aggregation function.

3.2.1.1 Inputs

The input to this function is the operation request, which is part of the message body sent by the DRS or user application program, when the Aggregator is activated.

The format of a join, outer join, union, or not-in-set request is the following:

```
{JOIN          rel1 rel2 APL rel1-rft rel2-rft result-rft
  OUTJOIN
  UNION
  NOT-IN-SET}
```

where:

rel1 = file name of the first operand.
rel2 = file name of the second operand.
APL = pointer to attribute pair list (APL)
 entry containing information about
 the join fields. This field is 0 for a
 union request.
rel1-rft = pointer to Result Field Table (RFT)
 entry containing field format
 information for rel1 for a join and outer
 join request. This field is 0 for a
 union request. For a not-in-set this
 field also represents the rft for the
 result.
rel2-rft = pointer to an RFT entry for rel2. This
 field is 0 for a union request.
result-rft = pointer to an RFT entry for the result
 table.

There is only one RFT pointer for a union request because the formats of both files and the result are identical. The requested operation of JOIN is represented by "4", UNION by "5", OUTER JOIN by "7" and NOT-IN-SET by "6".

The format of the APL is a list of join attribute pairs.

Each entry in the list contains the following:

rel1 attr1 rel2 attr2 link

where:

rel1 = not used
attr1 = Attribute Use Class number (AUC) of an
 attribute from rel1
rel2 = not used
attr2 = AUC from rel2
link = not used

Each entry in the RFT has the following format:

rel attr type size ND PID is-ptr

where:

rel = not used

attr = AUC of the field
type = field type (alphabetic, numeric, etc.)
size = number of bytes in the field
ND = number of decimal places of the field
PID = not used
is-ptr = not used

3.2.1.2 Processing

The aggregator inputs are passed to the main aggregator routine by an NTM message which is then unpacked into appropriate tables. These tables are then passed to the aggregator subroutine for processing. The aggregator subroutine also is passed these tables when initiated by a user application program.

3.2.2 FUNCTION AGG2: Perform Union

This function performs a Union on two like input operands and stores the result of the Union in a result file.

3.2.2.1 FUNCTION

The Union function appends one aggregator input file to the second aggregator input file creating a results file. The two input files must match in number, character types, and size for each field/columns. The starting routine for the main Union, or the calling routine for the in-line Union, will insure that the first input file is the larger of the two. The Union input files will not be sorted.

3.2.2.2 CONFIGURATION

3.2.2.2.1 MAIN UNION MODULE

The Union aggregator function is performed as a stand alone function when started by the DRS with a message from the NTM. The main Union module will receive and unpack the message from the NTM to determine the two Union input files. The main Union module then calls the subroutine Union module which controls the processing of the Union aggregator.

3.2.2.2.2 SUBROUTINE UNION MODULE

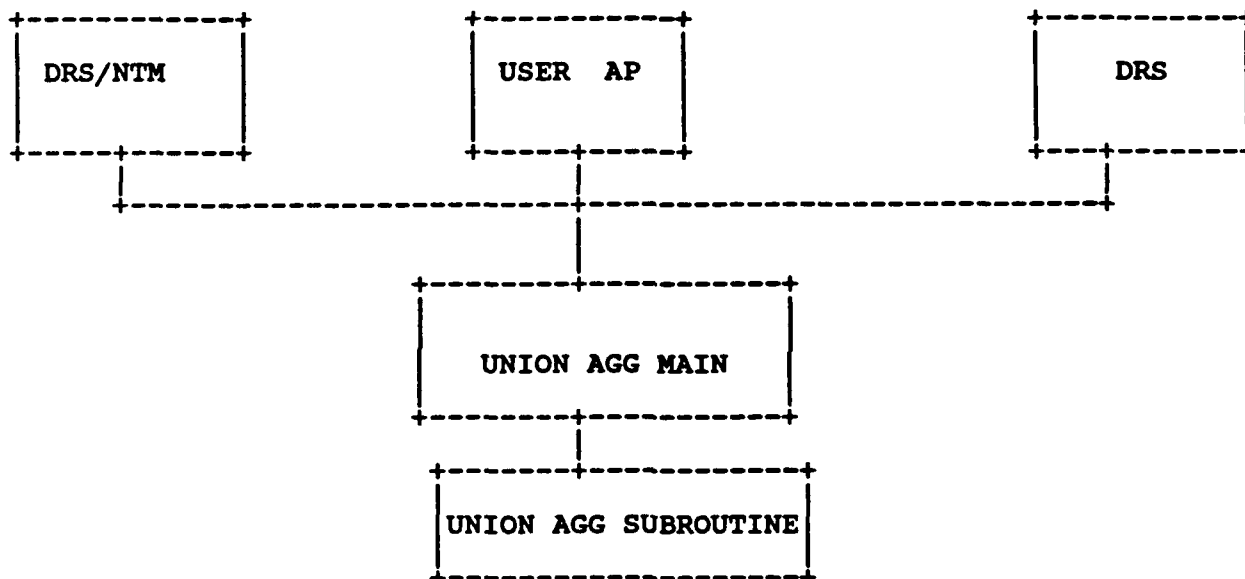
The subroutine Union module is called by the main Union routine, the DRS, or the user AP and is passed the names of the Union input files. The Union subroutine is the controlling

routine for all the Union aggregator logic and will return the results file name and status to the calling routine. No result record count is provided by the Union aggregator.

3.2.2.3 STANDARDIZATION

All aggregator functions have been standardized to facilitate maintenance and commonality among routines and functions. The Union aggregator does not use any common aggregator routines due to the limited amount of processing.

3.2.2.4 DIAGRAM/CHART - UNION AGGREGATOR



3.2.2.5 UNION INPUTS

The Union aggregator receives two input file names as input. The two input files must match in number, size, and character type for each column.

3.2.2.6 PROCESSING

3.2.2.6.1 UNION MAIN ROUTINE

The Union main routine is started by the DRS through an NTM message. The routine receives the message which contains the

name of the two Union input files. The main routine then calls the Union subroutine and passes it the name of the two input files.

3.2.2.6.2 UNION SUBROUTINE

The Union subroutine is the controlling routine for the Union aggregator logic. When called by the Union main, the DRS or a user AP, the Union subroutine performs an append of the second input file to the first input file. The DRS and/or the user AP will insure that the second input file is the smaller of the two input files. Upon completion, the first input file will become the results file.

3.2.2.6.3 PROCESSING LOGIC

Each record of the second input file is read and written to the first input file. This continues until end-of-file is reached. Both files are closed and the second input file is deleted.

The Union subroutine then passes the name of the results file to the calling program along with the program status.

The Union aggregator contains no special logic for processing of null values.

3.2.2.7 OUTPUTS

The Union output consists of the result of the appended input files. The first input file becomes the output results file and the name is passed to the calling routine with the program status.

3.2.2.8 ERROR HANDLING

The Union aggregator will use the standard "CDMP" error handling procedures with the addition of each module called having an identifying error message that indicates the error and where the error occurred.

3.2.3 FUNCTION AGG3: Perform Join

This function performs a Join on the input operands and stores the result in a specified file.

3.2.3.1 FUNCTION

The Join Aggregator performs a Join of two files based on the qualifications of the keys in each file. The result file is created from the Results Field Table (RFT) which could include

combinations of fields from each file. If the keys from the first file match the keys from the second file, then a record made from results RFT is written to the results file. The key fields are compared based on the qualifications of two fields with an operator. The legal relational operators are =, <, >, <=, >=, and !=. At least one of the operators in multiple key comparisons must be an "=".

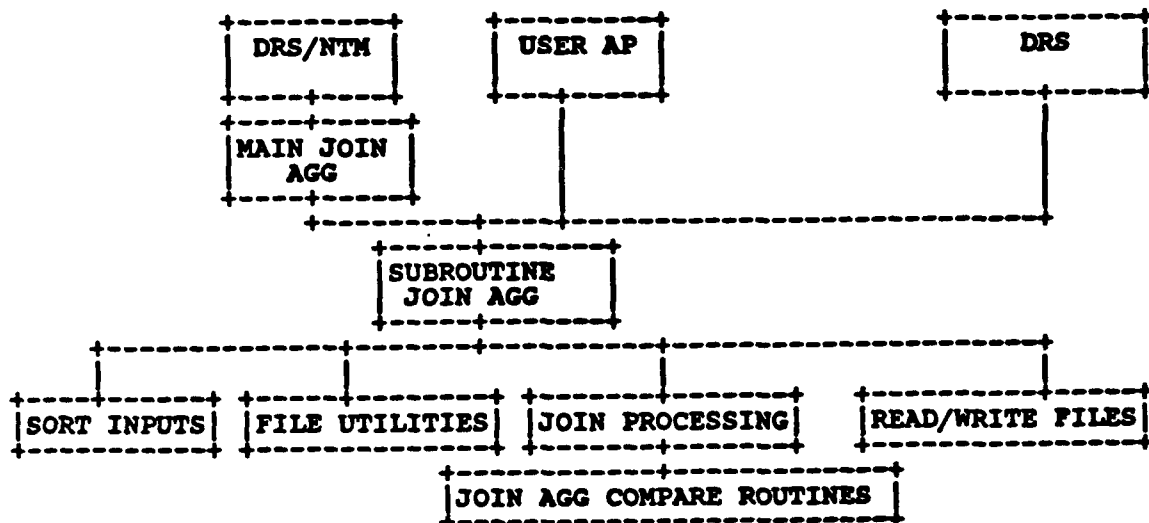
3.2.3.2 CONFIGURATION

The Join is configured as a Main Routine started by the NTM when called by the DRS, or as a Subroutine that can be called directly from the DRS for single node operations, or by a user AP for an intersect on a combined query. The Main Routine receives and unpacks the NTM messages containing the input files and descriptions, the Attribute Pair List (APL), and a Results Field Table (RFT) for each input and result file. The input files are the result of a user's Application Processor (AP) accessing a database and returning records that will participate in the Join. The subroutine is the controlling routine for the Join Logic and may be called by the main routine, the DRS, or the user's AP.

3.2.3.3 STANDARDIZATION

All Aggregator functions have been standardized to use the same file handling routines, sorting routines, and functions. This will aid in maintenance and commonality.

3.2.3.4 STRUCTURE CHART DIAGRAM



3.2.3.5 JOIN PROCESSING

The Join routine is called by the user's AP. This routine controls all Join processing and receives the input files, APL and RFT. It is used to communicate with the NTM and to unpack the NTM messages. The Join Subroutine, containing the join logic, is then called and the APL, RFT, and input file names are passed.

3.2.3.5.1 SUBROUTINE INPUTS

The Join Aggregator Subroutine is called by the Main Join Routine, the DRS, or a user's AP. The Join Subroutine receives the following inputs:

Input File 1
Input File 2
APL
RFT For Input File 1 (RFT)
RFT For Input File 2 (RFT2)
RFT For Results File (RFTTR)

ATTRIBUTE PAIR LIST (APL)

```
01 JQG-ATTRIBUTE-PAIR-LIST.
03   APL-MAX          PIC 99.
03   APL-USED         PIC 99.
03   APL-ROW-SIZE     PIC 99.
*
*   APL-ROW-SIZE IS NEEDED BY DRS FOR CALCULATING
*   THE TOTAL MESSAGE SIZE TO THE AGGREGATORS.
*
03 APL-ROW OCCURS 60 TIMES INDEXED BY APL-INDEX.
05   JQG-SUBTRANSL    PIC 999.
05   JQG-ATTRL        PIC 9(6).
05   JQG-SUBTRANSR    PIC 999.
05   JQG-ATTRR        PIC 9(6).
05   JQG-NEXT-PTR     PIC 99.
05   JQG-OP           PIC XX.
```

RFT

There is a separate RFT for each input file and the results file. The tables are identical except in name.

*

THE RESULT FIELD TABLES

01 RFT
03 RFT-MAX PIC 999 VALUE 200.
03 RFT-USED PIC 999 VALUE 0.
03 RFT-ROW-SIZE PIC 999 VALUE 23.
*
* THE ROW-SIZE IS NEEDED BY DRS TO DETERMINE THE
* TOTAL SIZE OF THE MESSAGE TO THE AGGREGATOR.
*
03 RFT-ENTRY OCCURS 200 TIMES
INDEXED BY RFT-INDEX.
05 RFT-PID PIC 9(6).
05 RFT-SUBTRANS PIC 999.
05 RFT-ATTR PIC 9(6).
05 RFT-SIZE PIC 999.
05 RFT-IS-PTR PIC 99.
05 RFT-TYPE PIC X.
05 RFT-ND PIC 99.

01 RFT2.
03 RFT2-MAX PIC 999 VALUE 200.
03 RFT2-USED PIC 999 VALUE 0.
03 RFT2-ROW-SIZE PIC 999 VALUE 23.
*
* THE ROW-SIZE IS NEEDED BY DRS TO DETERMINE THE
* TOTAL SIZE OF THE MESSAGE TO THE AGGREGATOR.
*
03 RFT2-ENTRY OCCURS 200 TIMES
INDEXED BY RFT2-INDEX.
05 RFT2-PID PIC 9(6).
05 RFT2-SUBTRANS PIC 999.
05 RFT2-ATTR PIC 9(6).
05 RFT2-SIZE PIC 999.
05 RFT2-IS-PTR PIC 99.
05 RFT2-TYPE PIC X.
05 RFT2-ND PIC 99.

01 RFTR.
03 RFTR-MAX PIC 999 VALUE 200.
03 RFTR-USED PIC 999 VALUE 0.
03 RFTR-ROW-SIZE PIC 999 VALUE 23.
*
* THE ROW-SIZE IS NEEDED BY DRS TO DETERMINE THE
* TOTAL SIZE OF THE MESSAGE TO THE AGGREGATOR.
*
03 RFTR-ENTRY OCCURS 200 TIMES
INDEXED BY RFTR-INDEX.
05 RFTR-PID PIC 9(6).
05 RFTR-SUBTRANS PIC 999.
05 RFTR-ATTR PIC 9(6).

05	RFTR-SIZE	PIC 999.
05	RFTR-IS-PTR	PIC 99.
05	RFTR-TYPE	PIC X.
05	RFTR-ND	PIC 99.

3.2.3.5.2 SORT JOIN OPERATORS (JQG-OP)

The Join operators in the APL are sorted to insure that an "=" operator exists and that all "=" operators are first on the APL. Use the following logic:

Unpack APL-ROW into the following table:

```
01  APL-WS-TABLE.
    03  APL-ROW-WS OCCURS 60 TIMES INDEXED BY APL-INDEX-WS.
        05  JQG-SUBTRANSL-WS      PIC 9(3).
        05  JQG-ATTRL-WS          PIC 9(6).
        05  JQG-SUBTRANSL-WS      PIC 9(3).
        05  JQG-ATTRL-WS          PIC 9(6).
        05  JQG-NEXT-PTR-WS       PIC 99.
        05  JQG-OP-WS             PIC XX.
```

This table will contain the sorted APL and will be used during comparison based on the key JQG-OP containing the value "=".

```
IF JQG-OP (APL-INDEX) contains the value "="
  MOVE APL-ROW (APL-INDEX) TO
    APL-ROW-WS (APL-INDEX-WS)
  UNTIL APL-INDEX > APL-USED.
```

Then, for all other operators: (>, <, >=, <=, !=):

```
IF JQG-OP(APL-INDEX) NOT EQUAL "="
  MOVE APL-ROW (APL-INDEX) TO APL-ROW-WS
    (APL-INDEX-WS)
  MOVE JQG-OP(APL-INDEX) TO CURRENT-APL-OPERATOR
IF OP-GT OR OP-LT OR OP-NE
  MOVE 1 TO APL-OPERATOR-STATUS
  UNTIL APL-INDEX > APL-USED.
```

3.2.3.5.3 RFT SWITCHING

Due to the possibility of having both local and remote subtransactions, RFT-SUBTRANS(1) is checked to see if it is equal to JQG-SUBTRANSL-WS(1). If it is not equal, the RFT and RFT2 tables need to be switched since they do not correspond to their matching subtransactions.

3.2.3.5.4 NAMFIL

Call the File I/O Primitive NAMFIL to retrieve a file name for the results file and two temporary files that will be used in the sort function.

3.2.3.5.5 FIND KEY ATTRIBUTES

Find the key attributes for each file and move the attribute description to the input file tables SORT-FILE1 and SORT-FILE2. The key attributes are found in the APL table and then compared to the RFT table to determine the attribute description. The JQG-ATTRL is compared to the RFT-ATTR until a match is found. This is repeated for the second input file with JQG-ATTRR compared to RFT-ATTR.

3.2.3.6.6 CREATE THE RESULTS FILE RFT

Compare RFTR-ATTR to each RFT-ATTR and RFT2-ATTR entry to determine in which file the ATTR resides. Maintain current record position to determine START-POS and keep count of size to determine record length. When a match is found

RFTR-ATTR (RFTR-INDEX) = RFT-ATTR (RFT-INDEX) or
RFT2-ATTR (RFT2-INDEX)

the corresponding table entries are moved to the results file table shown below:

```
01  FILER-DESC-TABLE.
03  RESULTS-FILE-RFT-DESC OCCURS 200 TIMES
      INDEXED BY RES-RFT-INDEX.
05  RES-FILE-NO          PIC 9.
05  RES-ATTR            PIC 9(9).
05  RES-SIZE            PIC 9(3).
05  RES-TYPE            PIC X.
05  RES-NO-DIG          PIC 99.
05  RES-START-POS       PIC 9(4).
05  RES-END-POS         PIC 9(4).
05  RES-NULL-POS        PIC 9.
```

This allows creation of the result file record for the records meeting the compare criteria.

3.2.3.6.7 SORT THE INPUT FILES

The two input files must be sorted in key order to facilitate any comparisons. The sort is performed by a File I/O Primitive. The routine is called and passed the input file names, the temporary file names that will receive the result of the sort, and the descriptions of each file's key fields.

3.2.3.6.8 OPEN AND READ FILES

Open input and result files. Read first record for both input files into buffers with a maximum record length of 2025 characters. Open and read are performed by File I/O Primitives.

3.2.3.6.9 SET MULTIPLE KEY CONDITIONS

Set up multiple key conditions based on number of APL-USED. Every key for each record will be compared based on the operator for each APL entry.

3.2.3.6.10 CREATE ATTRIBUTE BUFFERS

Read the sorted attribute descriptions from the RFT into input field descriptions for files 1 and 2. The left side buffer is for file 1 and the right side buffer is for file 2.

Set up the null conditions check. If either record key field is a null value, then the two key fields cannot be compared and the records cannot match. The null flags are indicated in each input file record buffer. Each record buffer is in the following format:

```
01    FILEx-BUFFER.  
      03    FILEx-REC-BUFFER      PIC X(2025)  
      03    NULL-FILEx-BUFFER REDEFINES FILEx-REC-BUFFER  
                                PIC X OCCURS 2025 TIMES  
                                INDEXED BY NFLx-INDX.
```

where x is either 1 or 2.

To determine the position of a field in the buffer for comparison, use an offset of RFT-USED + 1 as the record starting position.

The null buffer flag will indicate "0" for not NULL and "1" for Null.

3.2.3.6.11 CHECK FIELD CHARACTER TYPES

The only allowable type comparisons are character to character, numeric to numeric or signed, or signed to signed or numeric. Stop processing if there is a character type in one file and a non-character type in the other and display an error message.

3.2.3.6.12 CONVERSION AND COMPARISON OF FIELDS

The compare and conversion routines are called by the Join Subroutine. They are Fortran routines that convert the buffers to a standard data type, compare the values and set the join condition. If the field buffer character type is numeric or signed, the buffer must be converted to a real number for comparison. Signed overpunched numbers are converted to positive or negative digit by a Fortran routine according to the VAX conversion chart. These routines are VAX dependent.

After all conversion is completed, the two fields are compared to set the comparison condition. This condition is the output of the comparison routine.

Condition 1 - Left side is less than right side.
Condition 2 - Left side is equal to right side.
Condition 3 - Left side is greater than right side.
Condition 4 - Left side is equal to right side and
 left side is equal to previous left
 side.

3.2.3.6.13 MULTIPLE KEY PROCESSING

For multiple keys, if the condition matches the CURRENT-APL-OPERATOR (>, <, =, <=, >=, !=>) the next key field is compared until MULTI-KEY-IND > APL-USED. When all keys have been compared, the results are written or the record comparisons are terminated since the set of fields did not meet the condition. Another File1 or File2 record is then read.

3.2.3.6.14 JOIN PROCESSING

JOIN CONDITION OF 1

```
IF operator is "=" or ">" or ">=" (no match found)
  IF MATCH-CNT > 1
    backspace file2 MATCH-CNT times
    read file2
    read file1
    move 1 to MATCH-CNT
  ELSE
    read file1.

IF operator is "<" or "<=" or "!="
  IF more than 1 key and all keys not yet read
    INCREMENT KEY INDEX
  ELSE
    write results
    read file2
    IF EOF (file2)
      backspace file2 MATCH-CNT times
      read file2
      read file1
      move 1 to MATCH-CNT
    ELSE
      increment MATCH-CNT.
```

JOIN CONDITION OF 2

```
IF operator is "=" or ">=" or "<="
```



```
IF more than 1 key and all keys not yet read
  INCREMENT KEY-INDEX
ELSE
  write results
  read file2
  IF EOF (file2)
    backspace file2 MATCH-CNT times
  ELSE
    add 1 to MATCH-CNT.

IF operator is ">"
  IF MATCH-CNT > 1
    backspace file2 MATCH-CNT times
    read file2
    read file1
    move 1 to MATCH-CNT
  ELSE
    read file1.

IF operator is "<"
  read file2
  IF EOF (file2)
    read file1.

IF operator is "!="
  read file2
  IF EOF (file2)
    IF MATCH-CNT > 1
      backspace file2 MATCH-CNT times
      read file2
      read file1
      move 1 to MATCH-CNT
    ELSE
      read file1
  ELSE
    add 1 to MATCH-CNT.
```

JOIN CONDITION OF 3

```
IF operator is "<" or "<=" or "="
  read file2
  IF EOF (file2)
    read file1.

IF operator is "!=" or ">" or ">="
  IF more than 1 key and all keys not yet read
    increment KEY-INDEX
  ELSE
    write results
    read file2
    IF EOF (file2)
      backspace file2 MATCH-CNT times
      read file2
      read file1
      move 1 to MATCH-CNT
    ELSE
      increment MATCH-CNT.
```

JOIN CONDITION OF 4

```

IF operator is "=" or ">=" or "<="
  IF more than 1 key and all keys not yet read
    increment KEY-INDEX
  ELSE
    write results
    read file2
    IF EOF (file2)
      backspace file2 MATCH-CNT times
    ELSE
      increment MATCH-CNT.

```

```

IF operator is ">" or "<"
  read file1.
IF operator is "!="
  read file2
  IF EOF (file2)
    backspace file2 MATCH-CNT times
    read file2
    read file1
    move 1 to MATCH-CNT
  ELSE
    increment MATCH-CNT.

```

<CURRENT-APPE-OPERATOR>

	OP-GT >	OP-LT <	OP-EQ =	OP-LE <=	OP-GE >=	OP-NE !=
CND1	READ FILE1	MATCH FOUND WRITE RESULTS READ FILE2	READ FILE1	MATCH FOUND WRITE RESULTS READ FILE2	READ FILE1	MATCH FOUND WRITE RESULTS READ FILE2
CND2	READ FILE1	READ FILE2	MATCH FOUND WRITE RESULTS READ FILE2	MATCH FOUND WRITE RESULTS READ FILE2	MATCH FOUND WRITE RESULTS READ FILE2	READ FILE2
CND3	MATCH FOUND WRITE RESULTS READ FILE2	READ FILE2	READ FILE2	READ FILE2	MATCH FOUND WRITE RESULTS READ FILE2	MATCH FOUND WRITE RESULTS READ FILE2
CND4	READ FILE1	READ FILE1	MATCH FOUND WRITE RESULTS READ FILE2	MATCH FOUND WRITE RESULTS READ FILE2	MATCH FOUND WRITE RESULTS READ FILE2	READ FILE2

3.2.3.6.15 MATCHING RECORD PROCESSING

Each instance of matching keys in File1 will be repeated for each and every instance of matching keys in File2.

EXAMPLE:

This query:

```
Select 1.A, 1.B, 1.C, 2.B, 2.C from
A, B
where

1.A = 2.A and
1.B <= 2.B;
```

File 1			File 2		
A	B	C	A	B	C
A	1	X	A	1	RR
A	1	Y	A	1	RY
A	1	Z	B	1	RX
B	1	X	B	2	RZ
B	2	Y			

Produces this results file:

1.A	1.B	1.C	2.B	2.C
A	1	X	1	RR
A	1	X	2	RY
A	1	Y	1	RR
A	1	Y	2	RY
A	1	Z	1	RR
A	1	Z	2	RY
B	1	X	1	RX
B	1	X	2	RZ
B	2	Y	2	RZ

3.2.3.6.16 BACKSPACE LOGIC

The BACKSPACE logic for rewinding File2 requires that File2 be backspaced for each record which matches a File1 record. MATCH-CNT keeps a running total of the number of records to backspace for current File1 record. This process uses the File I/O Primitive SEKFIL to backspace, performing it MATCH-CNT times. Use of SEKFIL to obtain records that match the current File1 record is invoked when:

1. The current File1 record key is less than the current File2 record key (Condition 1) and MATCH-CNT is greater than 1 or File2 is EOF.

2. The current File1 record key is equal to the current File2 record key (Condition 2) and (CURRENT-APL-OPERATOR is ">" and MATCH-CNT is greater than 1) OR (CURRENT-APL-OPERATOR is ">=", "<=", "=", OR "!=" and File2 is EOF).

3. The current File1 record key is greater than current File2 record key (Condition 3) and (CURRENT-APL-OPERATOR is ">", ">=", or "!=" and File2 is EOF).

4. The current File1 record key is equal to previous File1 record key and current File1 record key is equal to current File2 record key (Condition 4) and (CURRENT-APL-OPERATOR is ">=", "<=", "=", or "!=" AND File2 is EOF).

The counter used to backspace the current File2 record is cleared after the backspace process has been completed for that record.

3.2.3.6.17 CREATE THE RESULTS FILE

First, the null flags, one for each attribute written to the results file, must be placed at the beginning of each record. Next, the File1 and File2 fields are put into the results file buffer using RFTR-INDEX. This is then written to the results file, and the number of records output is incremented.

3.2.3.6.18 JOIN PROCESSING LOGIC

The Join Subroutine will stop processing after all records in file 1 have been read and compared. The open files are all closed, the temporary input files are deleted and the name of the results file, number of results records, and the program status are passed back to the main routine or the calling routine.

The Main Join Routine receives the results file name, the results file record count, and the return status. The main routine then calls NSEND to send a message to the NTM giving the results file name, count, and status. To shut down the routine, an NTM Message is sent via TRMNAT that tells the NTM to terminate the Main Join Routine.

3.2.3.7 OUTPUT

The Join Aggregator will return the results file name, the results record count, and the program status. The results file can be a combination of the two input files, the format and record layout of which are found in the result file RFT.

3.2.3.8 ERROR HANDLING

The Join Aggregator uses the standard CDMF error handling procedures. In addition, each module called will have

an error message/code to identify the error and indicate where the error occurred.

3.2.4 FUNCTION AGG4: Perform Outer Join

This function performs an Outer Join operator on two input operands and stores the results in a specified file.

3.2.4.1 FUNCTION

The Outer Join Aggregator joins two files based on an Outer Join qualification on the keys in each file. The Outer Join is an extension of a join in which rows in the first file having no match in the second file appear in the result file with null values for any fields in the second file.

```
Select A.1, A.2, B.1, B.2
From   A,   B
Where  A.1 U= B.1;
```

can be logically stated as

```
if A.1 = B.1
    WRITE A.1, A.2, B.1, B.2
Else
    WRITE A.1, A.2, NULL, NULL.
```

EXAMPLE:

```
Select A.Name, A.No, B.Name, B.No
From   DEPT A, PROJECT B
Where  A.Name U= B.Name
```

<u>NAME</u>	<u>NO</u>
BLACK	99
BLUE	13
BROWN	12
GREEN	18
JONES	14
SMITH	16

<u>NAME</u>	<u>NO</u>
BLACK	199
ELDAR	98
GREEN	18
HENRY	01
SMITH	16
ZORK	44

RESULT

<u>A.NAME</u>	<u>A.NO</u>	<u>B.NAME</u>	<u>B.NO</u>
BLACK	99	BLACK	199
BLUE	13	(null)	(null)
BROWN	12	(null)	(null)
GREEN	18	GREEN	18
JONES	14	(null)	(null)
SMITH	16	SMITH	16

Only another Outer Join operator may be used with an Outer Join operator.

Where A.Name U= N.Name and
A.NO U= B.NO;

would yield

<u>RESULT</u>			
<u>A.NAME</u>	<u>A.NO</u>	<u>B.NAME</u>	<u>B.NO</u>
BLACK	99	(null)	(null)
BLUE	13	(null)	(null)
BROWN	12	(null)	(null)
GREEN	18	GREEN	18
JONES	14	(null)	(null)
SMITH	16	SMITH	16

3.2.4.2 CONFIGURATION

The Main Outer Join Routine is activated by the NTM when requested by the DRS. The Main Outer Join Routine then receives and unpacks an NTM message containing the input files and descriptions, the Attribute Pair List (APL), and a Results Field table (RFT) for each input and results file. The Outer Join Subroutine is then called and the APL, RFT, and input file names are passed. The Main Outer Join Routine is used to communicate with the NTM and to unpack the NTM messages. The Outer Join Subroutine contains the Outer Join processing logic.

3.2.4.3 OUTER JOIN PROCESSING

3.2.4.3.1 SUBROUTINE INPUTS

The Outer Join Aggregator Subroutine is called by the Main Outer Join routine, the DRS, or a user's AP. The Outer Join Subroutine receives the following inputs:

Input File 1
Input File 2
APL
RFT For Input File 1 (RFT)
RFT For Input File 2 (RFT2)
RFT For Results File (RFTR)

ATTRIBUTE PAIR LIST (APL)

01 JQG-ATTRIBUTE-PAIR-LIST.
03 APL-MAX PIC 99.
03 APL-USED PIC 99.
03 APL-ROW-SIZE PIC 99.

*
* APL-ROW-SIZE IS NEEDED BY DRS FOR CALCULATING
* THE TOTAL MESSAGE SIZE TO THE AGGREGATORS.

03 APL-ROW OCCURS 60 TIMES INDEXED BY APL-INDEX.

05	JQG-SUBTRANSL	PIC 999.
05	JQG-ATTRL	PIC 9(6).
05	JQG-SUBTRANSR	PIC 999.
05	JQG-ATTRR	PIC 9(6).
05	JQG-NEXT-PTR	PIC 99.
05	JQG-OP	PIC XX.

RFT

There is a separate RFT for each input file and the results file. The tables are identical except in name.

*

THE RESULT FIELD TABLES

01 RFT.

03	RFT-MAX	PIC 999 VALUE 200.
03	RFT-USED	PIC 999 VALUE 0.
03	RFT-ROW-SIZE	PIC 999 VALUE 23.

*

*

*

*

THE ROW-SIZE IS NEEDED BY DRS TO DETERMINE THE TOTAL SIZE OF THE MESSAGE TO THE AGGREGATOR.

03 RFT-ENTRY OCCURS 200 TIMES
INDEXED BY RFT-INDEX.

05	RFT-PID	PIC 9(6).
05	RFT-SUBTRANS	PIC 999.
05	RFT-ATTR	PIC 9(6).
05	RFT-SIZE	PIC 999.
05	RFT-IS-PTR	PIC 99.
05	RFT-TYPE	PIC X.
05	RFT-ND	PIC 99.

01 RFT2.

03	RFT2-MAX	PIC 999 VALUE 200.
03	RFT2-USED	PIC 999 VALUE 0.
03	RFT2-ROW-SIZE	PIC 999 VALUE 23.

*

*

*

*

THE ROW-SIZE IS NEEDED BY DRS TO DETERMINE THE TOTAL SIZE OF THE MESSAGE TO THE AGGREGATOR.

03 RFT2-ENTRY OCCURS 200 TIMES
INDEXED BY RFT2-INDEX.

05	RFT2-PID	PIC 9(6).
05	RFT2-SUBTRANS	PIC 999.
05	RFT2-ATTR	PIC 9(6).
05	RFT2-SIZE	PIC 999.
05	RFT2-IS-PTR	PIC 99.
05	RFT2-TYPE	PIC X.
05	RFT2-ND	PIC 99.

```
01  RFTR.
    03  RFTR-MAX          PIC 999 VALUE 200.
    03  RFTR-USED        PIC 999 VALUE 0.
    03  RFTR-ROW-SIZE    PIC 999 VALUE 23.
*
*  THE ROW-SIZE IS NEEDED BY DRS TO DETERMINE THE
*  TOTAL SIZE OF THE MESSAGE TO THE AGGREGATOR.
*
    03  RFTR-ENTRY OCCURS 200 TIMES
        INDEXED BY RFTR-INDEX.
        05  RFTR-PID      PIC 9(6).
        05  RFTR-SUBTRANS PIC 999.
        05  RFTR-ATTR     PIC 9(6).
        05  RFTR-SIZE     PIC 999.
        05  RFTR-IS-PTR   PIC 99.
        05  RFTR-TYPE     PIC X.
        05  RFTR-ND       PIC 99.
```

3.2.4.3.2 RFT SWITCHING

Due to the possibility of having both local and remote subtransactions, RFT-SUBTRANS(1) is checked to see if it is equal to JQG-SUBTRANSL-WS(1). If it is not equal, the RFT and RFT2 tables need to be switched since they do not correspond to their matching subtransactions.

3.2.4.3.3 NAMFIL

Call the File I/O Primitive NAMFIL to retrieve a file name for the results file and the two temporary files that will be used in the sort function.

3.2.4.3.4 SORT JOIN OPERATORS (JQG-OP)

Search the Outer Join operators in the APL to insure that an "U=" operator exists. All operators must be "U=", since only another Outer Join operator may be used with an Outer Join operator.

3.2.4.3.5 FIND KEY ATTRIBUTES

Find the key attributes for each file and move the attribute description to the input file tables SORT-FILE1 and SORT-FILE2. The key attributes are found in the APL table and then compared to the RFT table to determine the attribute description. The JQG-ATTRL is compared to the RFT-ATTR until a match is found. This is repeated for the second input file with JQG-ATTRR compared to RFT-ATTR.

3.2.4.3.6 CREATE THE RESULTS FILE

Compare the results file ATTR (RFTR-ATTR) to each RFT-ATTR and RFT2-ATTR entry to determine in which file the ATTR resides. Maintain current record position to determine START-POS and keep

count of size to determine record length. This allows creation of the result file record for the records meeting the compare criteria.

3.2.4.3.7 SORT THE INPUT FILES

The two input files must be sorted in key order to facilitate any comparisons. The sort is performed by a File I/O Primitive. The routine is called and passed the input file names, the temporary file names that will receive the result of the sort (to become the input files), and the descriptions of each file key fields.

3.2.4.3.8 OPEN AND READ FILES

Open input and result files. Read first record for both input files into buffers with a maximum record length of 2025 characters. Open and read are performed by File I/O Primitives.

3.2.4.3.9 SET MULTIPLE KEY CONDITIONS

Set up multiple key conditions based on number of APL-USED. Every key for each record will be compared based on the Outer Join Operator for each APL entry.

3.2.4.3.10 CREATE ATTRIBUTE BUFFERS

Read the sorted attribute descriptions from the RFT into input field descriptions for files 1 and 2. The left side buffer is for file 1 and the right side buffer is for file 2.

Set up the null conditions check. If either record key field is a null value, the two key fields cannot be compared and the records cannot match. The null flags are indicated in each input file record buffer. Each record buffer is in the following format:

```
01      FILEx-BUFFER.  
03      FILEx-REC-BUFFER    PIC X(2025).  
03      NULL-FILEx-BUFFER  REDEFINES FILEx-REC-BUFFER  
                                PIC X OCCURS 2025 TIMES  
                                INDEXED BY NLFx-INDX.
```

where x is either 1 or 2.

To determine the position of a field in the buffer for comparison, use an offset of RFT-USED + 1 as the record starting position.

The null buffer flag will indicate "0" for not NULL and "1" for Null.

```
If a file1 record key field is null
  Set OTJ-CONDITION to 1
  Write results record
  Read file 1

If a file2 record key field is null
  If EOF (file 2)
    Set OTJ-CONDITION to 1
    Write results record
    Read file 1
  ELSE
    Set OTJ-CONDITION to 3
    Read file 2
    If EOF (file2)
      Set OTJ-CONDITION to 1
      Write results record
      Read file 1.
```

3.2.4.3.11 CHECK FIELD CHARACTER TYPES

The only allowable type comparisons are character to character, numeric to numeric or signed, or signed to signed or numeric. Stop processing if there is a character type in one file and a non-character type in the other and display an error message.

3.2.4.3.12 CONVERSION AND COMPARISON OF FIELDS

The compare and conversion routines are called by the Outer Join Subroutine. These are Fortran routines that convert the buffers to a standard data type, compare the values and set the Outer Join condition. If the field buffer character type is numeric or signed, then the buffer must be converted to a real number for comparison. Signed overpunched numbers are converted to positive or negative digit by a Fortran routine according to the VAX conversion chart. These routines are VAX dependent.

After all conversion is completed, the two fields are compared to set the comparison condition. This condition is the output of the comparison routine.

```
Condition 1 - Left side is less than right side.
Condition 2 - Left side is equal to right side.
Condition 3 - Left side is greater than right side.
Condition 4 - Left side is equal to right side and
              left side is equal to previous left side.
```

3.2.4.3.13 MULTIPLE KEY PROCESSING

For multiple keys, if the condition matches the APL-OP VALUE (U=), the next key field is compared until MULTI-KEY-IND > APL-USED. When all keys have been compared, the results are written or the record comparisons are terminated since the set of fields did not meet the condition. Another File1 or File2 record is then read.

3.2.4.3.14

OUTER JOIN PROCESSING

OUTER JOIN CONDITION OF 1

```
IF MATCH-CNT > 1
  Backspace File2 MATCH-CNT times
  Read File1
  Read File2
  Move 1 to MATCH-CNT
ELSE
  Write results with nulls for File2 (no match)
  Read File1.
```

OUTER JOIN CONDITION OF 2

```
IF more than 1 key and all keys not yet read
  increment the key index
ELSE
  Write results (match found)
  Read File2
  IF EOF (File2)
    Backspace File2 MATCH-CNT times
    Read File2
    Read File1
    Move 1 to MATCH-CNT
  ELSE
    increment MATCH-CNT.
```

OUTER JOIN CONDITION OF 3

```
IF EOF (File2)
  Set OTJ-CONDITION to 1
  Repeat until EOF (File1):
    Write results with nulls for File2 (no match)
    Read File1
ELSE
  Read File2
  IF EOF (File2)
    Move 1 to OTJ-CONDITION
  Repeat until EOF (File1):
    Write results with nulls for File2 (no match)
    Read File1.
```

OUTER JOIN CONDITION OF 4

```
IF more than 1 key and all keys not yet read
  increment key index
ELSE
  Write results (match found)
  Read File2
  IF EOF (File2)
    Backspace File2 MATCH-CNT times
    Read File2
    Read File1
    Move 1 to MATCH-CNT
  ELSE
    Add 1 to MATCH-CNT.
```

3.2.4.3.15 MATCHING RECORD PROCESSING

Each instance of matching keys in File1 will be repeated for each and every instance of matching keys in File2.

EXAMPLE:

This query:

Select 1.A, 1.B, 1.C, 2.B, 2.C from A, B

Where

1.A U= 2.A and
1.B U= 2.B ;

FILE 1

<u>A</u>	<u>B</u>	<u>C</u>
A	1	X
A	1	Y
A	1	Z
B	1	X
B	1	Y

FILE 2

<u>A</u>	<u>B</u>	<u>C</u>
A	1	RR
A	1	RY
B	1	RX

Produces this result file:

<u>1.A</u>	<u>1.B</u>	<u>1.C</u>	<u>2.B</u>	<u>2.C</u>
A	1	X	1	RR
A	1	X	1	RY
A	1	Y	1	RR
A	1	Y	1	RY
A	1	Z	1	RR
A	1	Z	1	RY
B	1	X	1	RX
B	1	Y	1	RX

3.2.4.3.16 BACKSPACE LOGIC

The logic for rewinding input File2 requires that File2 be backspaced for each record that matches a File1 record. MATCH-CNT keeps a running total of the number of records to backspace for current File1 record. This process uses the File I/O Primitive SEKFIL to backspace, performing it MATCH-CNT times. Use of SEKFIL to obtain records that match the current File1 record is invoked when:

1. The current File1 record key is less than the current File2 record key (Condition 1) and MATCH-CNT is greater than 1.
2. The current File1 record key is equal to the current File2 record key (Condition 2) AND File2 is EOF.

3. The current File1 record key is equal to the current File2 record key and is also equal to the previous File1 record key (Condition 4) and File2 is EOF.

The counter used to backspace the current File2 record is cleared after the backspace process has been completed for that record.

3.2.4.3.17 CREATE THE RESULTS FILE

First, the null flags, one for each attribute written to the results file, must be placed at the beginning of each record. Next, the File1 and File2 fields are put into the results file buffer using RFTR-INDEX. If the OTJ-CONDITION is equal to 1, null values will be moved into the results file buffer in place of the File2 values. This buffer is then written to the results file, and the number of records output is incremented.

3.2.4.3.18 OUTER JOIN PROCESSING LOGIC

The Outer Join Subroutine will stop processing after all records in File1 have been read and compared. The open files are all closed, the temporary input files are deleted and the name of the results file, number of results records, and the program status are passed back to the main routine or the calling routine.

The Main Outer Join Routine receives the results file name, the results file record count, and the return status. The main routine then calls NSEND to send a message to the NTM giving the results file name, count, and status. To shut down the routine, an NTM Message is sent via TRMNAT to tell the NTM to terminate the Main Outer Join Routine.

3.2.4.4 OUTPUT

The Outer Join Aggregator will return the results file name, the results record count, and the program status. The results file can be a combination of the two input files; the format and record layout of which are found in the result file RFT.

3.2.4.5 ERROR HANDLING

The Outer Join Aggregator uses the standard CDMP error handling procedures. In addition, each module called will have an error message/code to identify the error and indicate where the error occurred.

3.2.5 FUNCTION AGG5: Perform Not-In-Set

This function performs a NOT-IN-SET (NIS) operation between two input files and stores the results in a specified file.

3.2.5.1 FUNCTION

The NIS Aggregator compares keys of two input files. Records from the first file whose keys do not match any records keys from the second file will be written to the results file.

EXAMPLE:

```
*#SELECT          :UVAR1
*#                :UVAR2
*#                :UVAR3
*#
*#
*#
*#
*#FROM
*#
*#              (SELECT A.COL1, A.COL2, A.COL3
*#                  FROM   TABLEA.A)
*#
*#DIFFERENCE
*#
*#              SELECT B.COL1, B.COL2, B.COL3
*#                  FROM   TABLEB.B)
*#ORDER BY UVAR2;
```

COL1	COL2	COL3
BLACK	12	PROD
BLACK	12	ENG
BLUE	14	ENG
BLUE	15	PROD
BROWN	16	MIS
GREEN	19	PROD
PURPLE	22	MIS
YELLOW	24	ENG

COL1	COL2	COL3
BLACK	12	ENG
BLUE	14	ENG
BROWN	16	MIS
GREEN	19	PROD
RED	23	PROD
YELLOW	24	PROD

RESULT

UVAR1	UVAR2	UVAR3
BLACK	12	PROD
BLUE	15	PROD
PURPLE	22	MIS
YELLOW	24	ENG

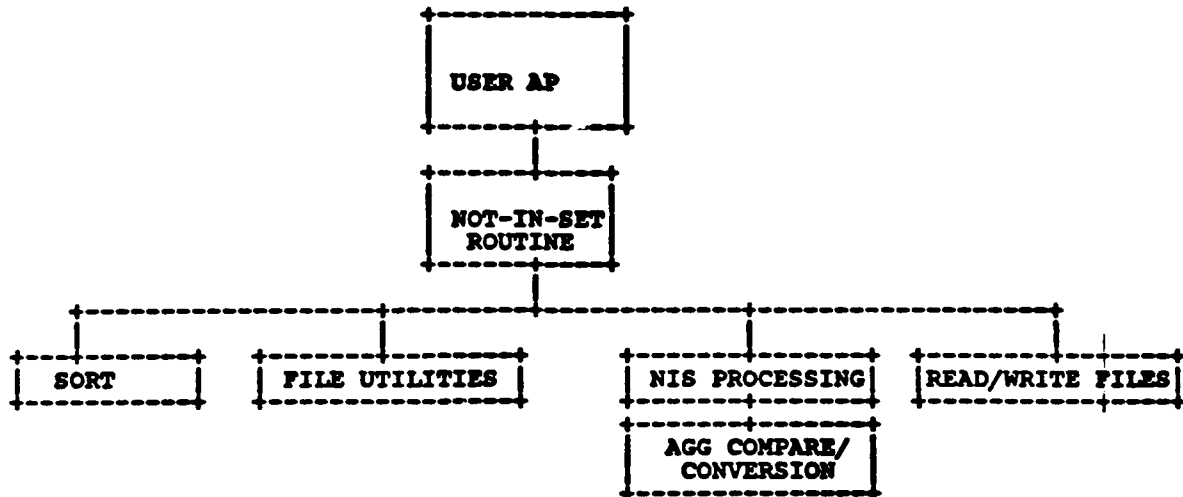
3.2.5.1.2 CONFIGURATION

The NIS Aggregator is used to perform the set operator "Difference" during a combined query and will only be called from the user's AP. The NIS Aggregator is no longer started by the NTM or DRS and only exists in the subroutine format.

3.2.5.3 STANDARIZATION

All Aggregator functions have been standardized to use the same file handling routines, sorting routines, and functions. This will aid in maintenance and commonality.

3.2.5.4 STRUCTURE CHART



3.2.5.5 NOT IN-SET PROCESSING

The NIS routine is called by the user's AP. This routine controls all NIS processing and receives the input files, APL, and RFT.

3.2.5.5.1 INPUTS

The NIS Aggregator receives the following inputs:

- Input File 1
- Input File 2
- APL
- RFT for Input File 1 (RFT)
- RFT for Input File 2 (RFT2)

These tables are then used to provide the descriptions of the files, records, and key fields.

ATTRIBUTE PAIR LIST (APL)

01 JQG-ATTRIBUTE-PAIR-LIST.

03 APL-MAX PIC 99.
03 APL-USED PIC 99.
03 APL-ROW-SIZE PIC 99.

*
*
*

APL-ROW-SIZE IS NEEDED BY DRS FOR CALCULATING
THE TOTAL MESSAGE SIZE TO THE AGGREGATORS.

03 APL-ROW OCCURS 60 TIMES INDEXED BY APL-INDEX.

05 JQG-SUBTRANSL PIC 999.
05 JQG-ATTRL PIC 9(6).
05 JQG-SUBTRANSR PIC 999.
05 JQG-ATTRR PIC 9(6).
05 JQG-NEXT-PTR PIC 99.
05 JQG-OP PIC XX.

RFT

There is a separate RFT for each input file and the results
file. The tables are identical except in name.

*

THE RESULT FIELD TABLES

01 RFT.

03 RFT-MAX PIC 999 VALUE 200.
03 RFT-USED PIC 999 VALUE 0.
03 RFT-ROW-SIZE PIC 999 VALUE 23.

*
*
*
*

THE ROW-SIZE IS NEEDED BY DRS TO DETERMINE THE
TOTAL SIZE OF THE MESSAGE TO THE AGGREGATOR.

03 RFT-ENTRY OCCURS 200 TIMES
INDEXED BY RFT-INDEX.

05 RFT-PID PIC 9(6).
05 RFT-SUBTRANS PIC 999.
05 RFT-ATTR PIC 9(6).
05 RFT-SIZE PIC 999.
05 RFT-IS-PTR PIC 99.
05 RFT-TYPE PIC X.
05 RFT-ND PIC 99.

01 RFT2.

03 RFT2-MAX PIC 999 VALUE 200.
03 RFT2-USED PIC 999 VALUE 0.
03 RFT2-ROW-SIZE PIC 999 VALUE 23.

*
*
*
*

THE ROW-SIZE IS NEEDED BY DRS TO DETERMINE THE
TOTAL SIZE OF THE MESSAGE TO THE AGGREGATOR.

03 RFT2-ENTRY OCCURS 200 TIMES
INDEXED BY RFT2-INDEX.
05 RFT2-PID PIC 9(6).
05 RFT2-SUBTRANS PIC 999.
05 RFT2-ATTR PIC 9(6).
05 RFT2-SIZE PIC 999.
05 RFT2-IS-PTR PIC 99.
05 RFT2-TYPE PIC X.
05 RFT2-ND PIC 99.

01 RFTR.
03 RFTR-MAX PIC 999 VALUE 200.
03 RFTR-USED PIC 999 VALUE 0.
03 RFTR-ROW-SIZE PIC 999 VALUE 23.

*
* THE ROW-SIZE IS NEEDED BY DRS TO DETERMINE THE
* TOTAL SIZE OF THE MESSAGE TO THE AGGREGATOR.
*

03 RFTR-ENTRY OCCURS 200 TIMES
INDEXED BY RFTR-INDEX.
05 RFTR-PID PIC 9(6).
05 RFTR-SUBTRANS PIC 999.
05 RFTR-ATTR PIC 9(6).
05 RFTR-SIZE PIC 999.
05 RFTR-IS-PTR PIC 99.
05 RFTR-TYPE PIC X.
05 RFTR-ND PIC 99.

3.2.5.5.2 RFT SWITCHING

Due to the possibility of having both local and remote subtransactions, RFT-SUBTRANS(1) is checked to see if it is equal to JQG-SUBTRANS(1). If it is not equal, the RFT and RFT2 tables need to be switched since they do not correspond to their matching subtransactions.

3.2.5.5.3 NAMFIL

Call the File I/O Primitive NAMFIL to retrieve a file name for the results file and the two temporary files that will be used in the sort function.

3.2.5.5.4 FIND KEY ATTRIBUTES

Find the key attributes for each file and move the attribute description to the input file table SORT-FILE1 and SORT-FILE2. The key attributes are found in the APL table and then compared to the RFT table to determine the attribute description. The JQG-ATTRL is compared to the RFT-ATTR until a match is found. This is repeated for the second input file with JQG-ATTRR compared to RFT2-ATTR.

3.2.5.5.5 SORT THE INPUT FILES

The two input files must be sorted in key order to facilitate any comparisons. The sort is performed by a File I/O Primitive. The routine is called and passed the input file names, the temporary file names that will receive the result of the sort, and the descriptions of each file key fields.

3.2.5.5.6 OPEN AND READ FILES

Open input and result files. Read the first record from both input files into buffers of maximum record length of 2025 characters. Open and read are performed by File I/O Primitives.

3.2.5.5.7 SET MULTIPLE KEY CONDITIONS

Set up multiple key conditions based on number of APL-USED. Every key for each record will be compared based on the operator for each APL entry.

3.2.5.5.8 CREATE ATTRIBUTE BUFFERS

Read the sorted attribute descriptions from the RFT into input field descriptions for files 1 and 2. The left side buffer is for file 1 and the right side buffer is for file 2.

Set up the null conditions check. If either record key field is a null value, the two key fields cannot be compared and the records cannot match. The null flags are indicated in each input file record buffer. Each record buffer is in the following format:

```
01 FILEx-BUFFER.  
03 FILEx-REC-BUFFER PIC X(2025).  
03 NULL-FILEx-BUFFER REDEFINES FILEx-REC-BUFFER  
PIC X OCCURS 2025 TIMES  
INDEXED BY NLFx-INDX.
```

where x is either 1 or 2.

To determine the position of a field in the buffer for comparison, use an offset of RFT-USED + 1 as the record starting position.

The null buffer flag will indicate "0" for not NULL and "1" for Null.

```
If a File1 record key field is null:  
Write results  
Read File1
```

```
If a File2 record key field is null:  
Read File2  
IF EOF (File2)
```

Repeat until EOF (File1):
Write results
Read File1.

3.2.5.5.9 CHECK FIELD CHARACTER TYPES

The only allowable type comparisons are character to character, numeric to numeric or signed, or signed to signed or numeric. Stop processing if there is a character type in one file and a non-character type in the other and display an error message.

3.2.5.5.10 CONVERSION AND COMPARISON OF FIELDS

The compare and conversion routines are called by the NIS Subroutine. They are Fortran routines that convert the buffers to a standard data type, compare the values and set the NIS condition. If the field buffer character type is numeric or signed, the buffer must be converted to a real number for comparison. Signed overpunched numbers are converted to positive or negative digits by a Fortran routine according to the VAX conversion chart. These routines are VAX dependent.

After all conversion is completed, the two fields are compared to the comparison condition. This condition is the output of the comparison routine.

Condition 1 - Left side is less than right side.

Condition 2 - Left side is equal to right side.

Condition 3 - Left side is greater than right side.

Condition 4 - Left side is equal to right side and left side is equal to previous left side.

3.2.5.5.11 NIS PROCESSING

NIS CONDITION OF 1

Write results
Read File1

NIS CONDITION OF 2

IF more than 1 key and all keys not yet read
increment key index

ELSE
Read File1.

NIS CONDITION OF 3

IF EOF (File2)
Repeat until EOF (File1):

```
        Write results
        Read File1
ELSE
    Read File2
    IF EOF (File2)
        Repeat until EOF (File1):
            Write results
            Read File1.
```

NIS CONDITION OF 4

```
IF more than 1 key and all keys not yet read
    increment key index
ELSE
    Read File1.
```

3.2.5.5.12 CREATE THE RESULTS FILE

The results file consists of file1 records meeting the NIS criteria. The null flags for the file1 fields are written to the record as well.

3.2.5.5.13 NIS PROCESSING LOGIC

The NIS Subroutine will stop processing after all records in file1 have been read and compared. The open files are all closed, the temporary input files are deleted and the name of the results file, number of results records, and the program status are passed back to the calling routine.

3.2.5.6 OUTPUT

The NIS Aggregator will return the results file name, the results record count, and the program status to the calling routine.

3.2.5.7 ERROR HANDLING

The NIS Aggregator uses the standard CDMP error handling procedures. In addition, each module called will have an error message/code to identify the error and indicate where the error occurred.

3.3 Special Requirements

Principles of structured design and programming will be adhered to.

3.4 Human Performance

Not applicable.

3.5 Database Requirements

Not applicable.

3.6 Adaptation Requirements

The system will be implemented at the ICAM IISS Test Bed site located at the Arizona State University facility in Tempe, AZ. The Aggregator processes will be implemented on the VAX VMS host and the IBM host. The VAX and IBM versions will be the same except for the implementation of the system sort.

SECTION 4

QUALITY ASSURANCE PROVISION

In preparation for describing requirements for quality assurance provisions, it is appropriate to define the terms "test" and "debug" which are often used interchangeably. "Testing" is a systematic process that may be preplanned and explicitly scheduled. Test techniques and procedures may be defined in advance and a sequence of test steps may be specified. "Debugging" is the process of isolation and correction of the cause of an error. To start with, the concept of "antibugging" is recommended in the construction of the software modules. In his text on software development (Techniques of Program Structure and Design, Prentice-Hall, 1975) Yourdon defines antibugging as "the philosophy of writing programs in such a way as to make bugs less likely to occur, and when they do occur (which is inevitable), to make them more noticeable to the programmer and the user." That is, do as much error checking as is practical and possible in each routine.

Among the tests that should be incorporated into all software are:

1. input data checks
2. interface data checks, i.e., tests to determine validity of data passed from calling routine
3. database verification
4. operator command checks
5. output data checks

Not all tests are required in all routines, but error checking is an essential part of all software.

The CI quality assurance provisions must consist of three levels of test, validation and qualification of the constructed application software.

1. The initial level can consist of the normal testing techniques that are accomplished during the construction process. They consist of design and code walk-throughs, unit testing, and integration testing. These tests will be performed by the design team which will be organized in a manner similar to that discussed by Weinberg in his text on software development team organization (The Psychology of Computer Programming New York: Van Nostrand Reinhold, 1971). Essentially a team is assigned to work on a subsystem or CI. This approach has been referred to as "adaptive teams" and "egoless teams." Members of the team are involved in the overall

design of the subsystem; there is better control and members are exposed to each others' design. The specific advantage from a quality assurance point is the formalized critique of design walk-throughs which are a preventive measure for design errors and program "bugs." Structured design, design walk-throughs and the incorporation of "antibugging" facilitate this level of testing by exposing and addressing problem areas before they become coded "bugs."

2. Preliminary qualification tests of the CI are performed to highlight the special functions of the CI from an integrated point of view. Certain functional requirements may require the cooperative execution of one or more modules to achieve an intermediate or special function of the CI. Specific test plans will be provided for the validation of this type of functional requirement including preparation of appropriate test data. (Selected functions from 3.2 must be listed).
3. Formal Qualification Test will verify the functional performance of all the modules, within the CI as an integrated unit, that accept the specified input, perform the specified processes and deliver the specified outputs. Special consideration must be given to test data to ensure verification that proper interface of modules has been constructed.

SECTION 5

PREPARATION FOR DELIVERY

The implementation site for the constructed software will be the ICAM Integrated Support System (IISS) Test Bed site located at Arizona State University in Tempe, AZ. The required computer equipment will have been installed. The constructed software will be transferred to the IISS system via appropriate storage media.