WRDC-TR-90-8007
Volume V
Part 30

WRIGHT RESEARCH & DEVELOPMENT CENTER

INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume V - Common Data Model Subsystem
Part 30 - File Utilities 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

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

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

25 July 91
DATE

FOR THE COMMANDER:

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

25 July 91
DATE

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 620341330	5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-90-8007 Vol. V, Part 30
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	PROGRAM ELEMENT NO. 78011F · PROJECT NO. 595600 · TASK NO. F95600 · WORK UNIT NO. 20950607

12. PERSONAL AUTHOR(S)
Control Data Corporation: Apicella, M. L., Singh, S.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Yr.,Mo.,Day)*	15. PAGE COUNT
Final Report	4/1/ -12/30/90	1990 September 30	17

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 File Utility computer programs. These utilities provide file transfers, file deletes, and unique file naming services to other components of IISS.

BLOCK 11:

INTEGRATED INFORMATION SUPPORT SYSTEM
Vol V -Common Data Model Subsystem

Part 30 - File Utilities Development Specification

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson	22b. TELEPHONE NO. *(Include Area Code)* (513) 255-7371	22c. OFFICE SYMBOL WRDC/MTI

DD FORM 1473, 83 APR

EDITION OF 1 JAN 73 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

DS 620341330
30 September 1990

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:

SUBCONTRACTOR	ROLE
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

- To delete a file containing intermediate aggregation results.
- To delete a file containing the final aggregated results after it has been transformed from Conceptual to External format by the C/E transformer.
- To delete a file containing the final results from an NDML request after the information has been processed by the requesting application.
- To delete temporary files used by the NDML precompiler.

The primary uses of the File Namer are:

- To provide system wide unique file names for any requesting process.
- Specifically, unique file names are needed by the NDML precompiler for generated software:
  - request processor main routines
  - request processor subroutines
  - conceptual to external transform
  - conceptual level selection modules
- Specifically, unique file names are needed by the NDML precompiler for temporary files and for storage of generated programs.

The primary uses of the Module Namer are:

- To provide system wide unique software module names for a requesting process.
- Specifically, unique module names are needed by the NDML precompiler for all generated software:
  - request processor main routines
  - request processor subroutines
  - conceptual to external transforms
  - conceptual level selection modules

SECTION 2

DOCUMENTS

2.1 Reference Documents

1. DeJean, J.P., Test Bed System Development Specification, General Electric Company, Schenectady, New York, November 9, 1982.

SECTION 3

REQUIREMENTS

3.1 Computer Program Definition

3.1.1 System Capacities

The File Transfer operates on ASCII files with records that are fixed in length (all same size) or variable length record files that do not exceed a maximum; records exceeding the maximum are truncated.

The File Delete operates on ASCII files.

3.1.2 Interface Requirements

3.1.2.1 Interface Block Diagram

```
+----------+   FILE      +-------+ INITIATION     +---------+
|          |  TRANSFER   |       |  MESSAGE       |         |
|REQUESTING|  REQUEST    | FILE  |--------------->|  FILE   |
|          |------------>|       | ACKNOWLEDGE    |         |
| PROCESS  |             | SEND  |<---------------+ RECEIVE |
|          | COMPLETION  |       | DATA RECORDS   |         |
|          | & STATUS    |       |--------------->|         |
|          |<------------|       |  COMPLETION    |         |
|          |             |       |  & STATUS      |         |
|          |             |       |<---------------+         |
+----------+             +-------+                +---------+
```

The File Transfer CI is composed of two functional components, File Send and File Receive.

A requesting process sends a File Transfer message to the File Send at the host where the original file to be transmitted resides. The File Send then sends an initiation message to the File Receive at the host where the file is to be created. After receiving acknowledgment from File Receive the File Send reads the source file and creates messages containing the file data. It transmits these messages to the File Receive where they are reconstructed into a new file. When the file transfer is completed, File Send notifies the requesting process.

```
+------------+                           +--------------+
|            |    DELETE REQUEST         |              |
| REQUESTING |-----------------------> |     FILE     |
|            |                           |              |
|  PROCESS   |                           |    DELETE    |
|            |                           |              |
|            |                           |              |
+------------+                           +--------------+
```

A requesting process sends a File Delete request in the form of a message to the host where the file to be deleted resides.

```
+-----------+ FILE NAME REQUEST  +--------+  NEXT
|           |------------------->|        |  NAME   +------------+
|REQUESTING |                    |  FILE  |<--------|            |
|           |                    |        |         | PERMANENT  |
|  PROCESS  |                    | NAMER  |-------->|            |
|           |<-------------------|        |  LAST   |    FILE    |
|           |FILE NAMES & STATUS |        |  NAME   |            |
+-----------+                    +--------+  USED   +------------+
                                                    "LAST FILE
                                                     NAME USED"
```

A requesting process sends File Name request in the form of a message to the File Name queue server. After generating the next group of names, the File Namer sends a reply message with a status to the requesting process. The file "LAST FILE NAME USED" is used to periodically record the last file name assigned for each host.

```
+-------------+ MODULE NAME     +-------------+             +------------+
|             |    REQUEST      |             |NEXT NAME    |            |
|             |---------------->|             |<----------  |            |
| REQUESTING  | MODULE NAMES    |   MODULE    |LAST NAME    | PERMANENT  |
|  PROCESS    | AND STATUS      |   NAMER     |  USED       |    FILE    |
|             |<----------------|             |----------->|            |
+-------------+                 +-------------+             +------------+
                                                            "LAST MODULE
                                                             NAME USED"
```

A requesting process sends a Module Name request in the form of a message to the Module Name queue server. After generating the next group of names, the module namer sends a reply message with a status to the requesting process. The file "LAST MODULE NAME" is used to periodically record the last module name assigned.

3.2 Detailed Functional Requirements

The following subsections document the File Utility's major functions and describes the input-output requirement.

3.2.1 Functi›n FSD - File Send

The file send function transfers files from host to host in the testbed environment. The file send function residing at one host sends its contents to a file receive function on a remote host. The file send receives file characteristics and operating instructions from a requesting process. It, in turn, passes control information to the file receive function.

3.2.1.1 Inputs (from requesting process)

A. Name of file to be sent.
B. Record length of records contained in file.
C. Name of file to be built.
D. Logical location where file is to be sent.
E. Indicator on whether to delete or keep input file after transfer.
F. "Ready to Receive" (from File Receive).
G. "End of File Transfer" or "File Transfer Unsuccessful" (from File Receive).

3.2.1.2 Processing

A. Receive File Transfer initiation message from requesting process via the NTM, to transfer file. The message contains the inputs A thru E from 3.2.1.1, above.
B. Send message to location where file is to be sent to initiate File Receive routine. Send name of file to be created and record length from step B, above.
C. Receive "Ready to Receive" message from File Receive routine.
D. Open input file to be sent.
E. Read input records and place in output buffer. Continue until output buffer is full or all records from file have been read. Keep a count of total number of records in file.
F. Send message to File Receive routine containing data from E, above.
G. Repeat E and F, as necessary, to send entire file.
H. Send message to file receive routine inidcating all records have been sent and total record count from step E.
I. Wait for "End of File Transfer" or "File Transfer Unsuccessful" completion message from File Receive.

   1. If "File Transfer Unsuccessful" is received, check to see if this is first or second attempt to send the file. If first, try again by restarting from the beginning. If second try, send unsuccessful message to requesting process and terminate.

   2. If "End of File Transfer" continue with K, below.

K. Check input parameter E, under 3.2.1.1 above and delete original input file if the delete parameter is set. Note: use the File Delete utility to execute this step.

L. Send a completion message to the process requesting the file transfer.

M. Terminate the File Send.

3.2.1.3 Outputs

A. Message to receiving host to initiate File Receive.

- o Name of file to be created
- o Record length

B. Records containing data to be transferred.

C. "End of File" message (containing count of number of records sent).

D. Completion message to requesting process containing transfer status.

3.2.2 Function FRC - File Receive

The File Receive function operates in conjunction with the *File Send function in the test bed environment.* File Send, residing at one host, sends its contents to File Receive on a remote host. Prior to receiving any of the file contents, File Receive receives control information from File Send that contains the name of the file to be built and other file characteristics.

3.2.2.1 Inputs

A. Message from File Send to initiate receive process.

- o Name of file to be created.
- o Record length of records in the file.

B. Data records to build new file.

C. "End of File" message from File Send, which includes a count of the records sent.

3.2.2.2 Processing

A. Receive initiation message from File Send. (See input A in 3.2.1.1, above).

B. Create (open) output file to be written.

C. Send "Ready to Receive" message to File Send.

D. Read input records being sent by File Send and place on output file opened in B, above. Keep a count of number of logical records received and written.

E. Repeat Step D until all records have been written to output file as denoted by "End of File" message from File Send.

F. Compare count of number of logical records written against number of records sent. Then take one of the following actions:

- o If the counts are equal, send "End of File Transfer" message to File Receive.
- o If the counts are not equal, delete the output file being created and send "File Transfer Unsuccessf'l" message to the File Send routine.

G. Close the output file.

H. Terminate.

3.2.2.3 <u>Outputs</u>

A. "Ready to Receive" message to File Send.

B. Newly created data file.

C. "End of File Transfer" message or "File Transfer Unsuccessful" message

3.2.3 <u>Function FDL - File Delete</u>

The File Delete utility purges files residing on the local host. The request to delete a file can originate on any host in the IISS environment.

3.2.3.1 <u>Input</u>

A. File Delete initiation message containing name of file to be deleted.

3.2.3.2 <u>Processing</u>

A. An interface module, upon receiving the file name and its host of residence, will issue an operating system call to delete or purge the file if the file is on the same host as the interface module.

B. If the file is "off-host", a message is sent to the file delete process on that host which will then execute a call to the operating system (on local host) to delete the file using the file name supplied in input A, above.

C. No status reply is expected from the file delete interface or the file delete process.

3.2.4 Function FNM - File Namer

The File Namer utility generates unique file names for the requesting process.

3.2.4.1 Input

A. File Namer must be told by the requester that it needs a file name. The fact that a call is made to the interface module indicates this.

B. The interface module may send a message to the file name server for a group of file names. The fact that a requesting message arrives at the file name server indicates that a group of file names is desired.

3.2.4.2 Processing

A. The file name interface maintains a group of file names received from the server. When this group has been distriuted, one per request, the interface module will request another group from the server.

B. The file name server will maintain an in-memory table of the last name assigned for each host. This was loaded from the CDM table FILE_NAME_HOST at process initiation. When a request comes in, a block of names is generated by incrementing the numeric portion of the name. If the numeric portion "rolls" past all 9's, then the letter portion of the name is incremented. The group of file names is sent back to the requestor in the reply message.

C. This function assumes no other users of the host computer will be generating file names of the type developed, as no operating system calls or checks are made. This can be insured by reserving a special directory or file name logical for IISS use.

3.2.4.3 Output

A. A database table (FILE_NAME_HOST) of the next file names to be used is stored in the CDM database to prevent re-use of file names across system shutdown and startup.

B. A group of file names is sent in a reply message to the requesting file name interface routine.

C. A single file name is returned from the file name interface routine to the requesting application.

3.2.5 Function MNM - Module Namer

The Module Namer utility generates unique module names for the requesting process.

3.2.5.1 Input

A. Module Namer must be told by the requester that it needs a module name. The fact that a call is made to the interface module indicates this.

B. The interface module may send a message to the module name server for a group of module names. The fact that a requesting message arrives at the server indicates that a group of file names is desired.

3.2.5.2 Processing

A. The module name interface maintains a group of module names received from the module server. When this group has been distributed, one per request, the interface module will request another group from the queue server.

B. The module name queue server will maintain an in-memory table of the last name assigned. This was loaded from the CDM table NEXT_MOD_NAME at process initiation. When a request comes in, a block of names is generated by incrementing the numeric portion of the name. If the numeric portion "rolls" past all 9's, then the letter portion of the name is incremented. The group of module names is sent back to the requestor in the reply message.

C. This function assumes no other users of the host computer will be generating module names of the type developed, as no operating system calls or checks are made.

3.2.5.3 Output

A. A database table (NEXT_MOD_NAME) of the next module names to be used is stored in the CDM database to prevent re-use of module names across system shutdown and startup.

B. A group of module names is sent in a reply message to the requesting module name interface routine.

C. A single module name is returned from the module name interface routine to the requesting application.

SECTION 4

QUALITY ASSURANCE PROVISIONS

4.1 Introduction and Definitions

"Testing" is a systematic process that may be preplanned and explicitly stated. 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 and error.

4.2 Computer Programming Test and Evaluation

The quality assurance provisions for testing will 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.

The integation test developed for the file utilities will consist of a number of test cases developed for other components of the CDM. Because file utilities are essentially service modules, they will be tested by other CDM components.

Unit testing will primarily involve testing each of the utilities with skeleton routines calling the "service" interface module directly.

SECTION 5

PREPARATION FOR DELIVERY

The implementation site for the constructed software will be the ICAM Integrated Support System (IISS) Test Bed Site located in Tempe, Arizona. The software associated with the file utilities will be clearly identified and will include instructions on procedures to be followed for installation of the release.

☆U.S. GOVERNMENT PRINTING OFFICE: 1992 - 648-127/62320