PNL--7489

DE91 005215

SOLID WASTE PROJECTION MODEL: MODEL VERSION 1.0 TECHNICAL REFERENCE MANUAL

M. L. Wilkins
V. L. Crow
D. E. Buska
S. J. Ouderkirk[a]

November 1990

Prepared for
the U.S. Department of Energy
under Contract DE-AC06-76RLO 1830

Pacific Northwest Laboratory
Richland, Washington 99352

(a) Boeing Computer Services Richland, Richland, Washington.

EXECUTIVE SUMMARY

The Solid Waste Projection Model (SWPM) system is an analytical tool developed by Pacific Northwest Laboratory (PNL) for Westinghouse Hanford Company (WHC). The SWPM system provides a modeling and analysis environment that supports decisions in the process of evaluating various solid waste management alternatives.

This document, one of a series describing the SWPM system, contains detailed information regarding the software utilized in developing Version 1.0 of the modeling unit of SWPM. This document is intended for use by experienced software engineers and supports programming, code maintenance, and model enhancement.

Those interested in using SWPM should refer to the SWPM Model User's Guide. This document is available from either the PNL project manager (D. L. Stiles, 509-376-4154) or the WHC program monitor (B. C. Anderson, 509-373-2796).

CONTENTS

FIGURES

1.0 INTRODUCTION

The Solid Waste Projection Model (SWPM) system is an analysis tool developed by Pacific Northwest Laboratory (PNL) for the Solid Waste Technology Section of Westinghouse Hanford Company to address complex waste management issues. The SWPM system provides the ability to develop projections of solid waste volumes and characteristics and evaluate alternative waste treatment and disposal strategies.

A generic representation of the system modeled by SWPM is shown in Figure 1.1. Waste is received from waste generating facilities and is distributed to various operations. Operations are defined as either "treatments" or "disposals" and are linked to other operations to represent a given waste management scheme. Each operation has an associated storage option to track waste volumes that arrive in excess of defined capacity.

The SWPM system, shown in Figure 1.2, consists of three modules: the database, the user interface, and the model. The user interface controls access to the data libraries and operation of the model with a system of pull-down menus. The model consists of the algorithms to calculate treatment, storage, and disposal (TSD) volumes and costs, and the output report writer (ORW), which formats model results into readable reports. The model is

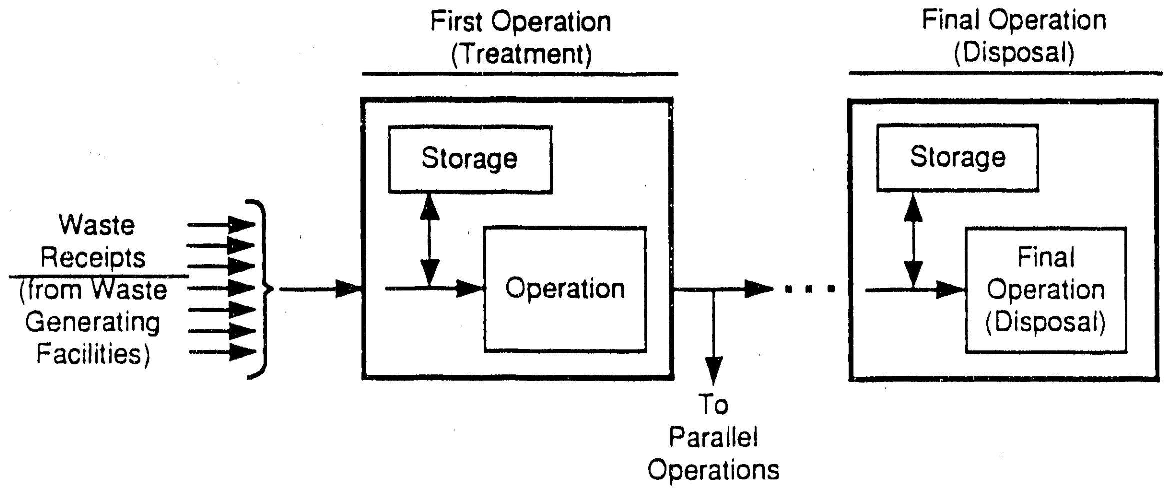

First Operation (Treatment)
Final Operation (Disposal)
Storage
Storage
Waste Receipts (from Waste Generating Facilities)
Operation
Final Operation (Disposal)
To Parallel Operations

FIGURE 1.1. The Waste Management System Representation

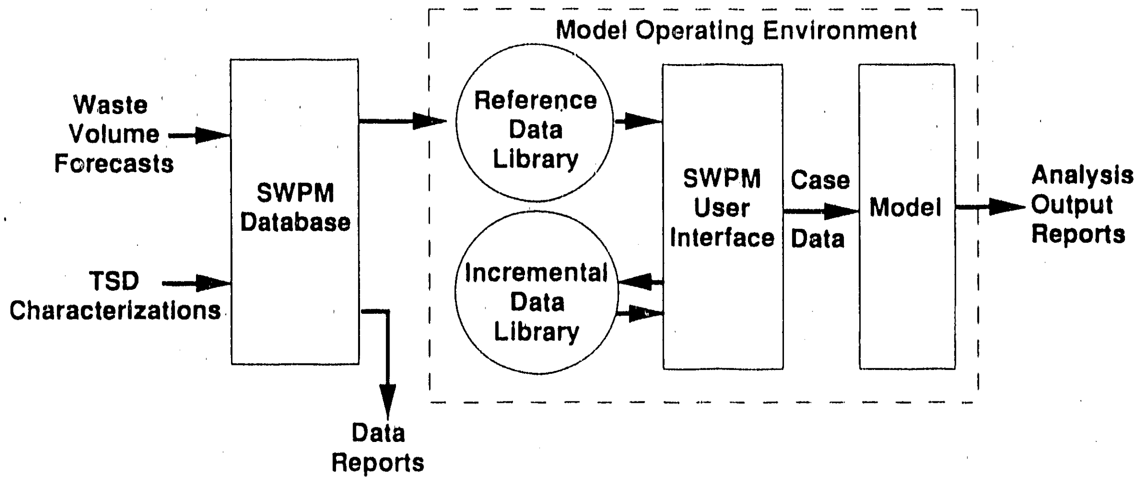


FIGURE 1.2. The SWPM System

supported by a database, which is used to store and maintain waste volume projections, as well as the description of the TSD operations and facilities. This data is transferred to the model via the specially formatted electronic Reference Data Library (RDL).

This SWPM Technical Reference Manual discusses the files and directory structure of the user interface, the projection file interface (PFI), the model, and the ORW. Together these packages comprise SWPM Version 1.0. Five additional documents complete the set of documentation that provide instructions in the use, maintenance, and application of the total SWPM system:

- System Overview - provides an overview of the SWPM system and an assessment of potential applications
- Database Users Guide - provides instructions for data entry, maintenance, and reporting
- Model User's Guide - provides instructions for model operation and execution
- Database Technical Reference Manual - describes database software, utilities, and structure
- System Administration Manual - provides instructions for long-term system administration and maintenance.

Copies of these documents or any further information may be obtained from the PNL project manager (D. L. Stiles, 509-376-4154) or from the WHC program monitor (B. C. Anderson, 509-373-2796).

2.0 SWPM USER INTERFACE

The key to operating SWPM is the graphical user interface, implemented in OS/2 Presentation Manager (PM), which accesses the RDL and guides the user through the selection of case-specific data sets. Once this case has been specified, a SWPM run may be initiated from the user interface. After completion of a SWPM run, the user interface also prompts the user to select from among several standard output tables, reporting SWPM results in a standard format. This chapter provides a detailed description of the operations performed by the user interface and the codes and routines which execute these operations.

2.1 USER INTERFACE DATA STRUCTURES

The C-language header file UI_LOAD.H defines the data structures that are common to the SWPM User Interface and the PFI routines that read and write SWPM data files. These data structures form the mechanism of communication between the user and SWPM data files.

The constants listed below define the maximum number of SWPM data objects that the user interface will accept in each entity list. These values are currently defined as:

- MAXGEN — Maximum number of waste generators; 400 objects.
- MAXCLASS — Maximum number of waste classes; 30 waste classes.
- MAXOPTYP — Maximum number of generic operation types; 200 objects.
- MAXOP — Maximum number of waste class-specific operations; set to be 2(MAXOPTYP*MAXCLASS) ÷ 5.
- MAXYEAR — Maximum number of years a scenario can run; 50 years.
- MAXGROUP — Maximum number of groups to which an object may belong; 20 groups.
- MAXGROUPALL — Maximum number of different group names for all objects; set to be 4(MAXGROUP).
- MAXSC — Maximum number of generic scenario names; 100 names.

- <u>MAXPS</u> Maximum number of object-specific scenario names; set to be (MAXOP + MAXGEN).
- <u>MAXNAM</u> Maximum number of characters in an object's name; 31 characters.
- <u>MAXCOM</u> Maximum number of characters in a single comment field; 255 characters.

While these constants define the limits of the user interface, the model is more constrained. **The total number of elements defined in a given list** (waste generators, operations, etc.) **may not exceed 255 for the model. Further, the total number of operations, plus waste generators, plus waste classes, may not exceed 900 elements.**

The object data structures used by the SWPM are described in the following listing.

```
typedef struct _SC {   Define the SC (Scenario) object.

      char  *name               Scenario name.  May be used with waste
                                generators and operations.

      BOOL  perm                TRUE if this scenario name was loaded from a
                                library; FALSE if the user created it during
                                this session.
} SC.

typedef struct _WC {   Define the WC (Waste Class) object (an array of these
                       objects will serve to enumerate waste classes selected
                       by the user.)

      char  *name               Name of waste class (e.g., "RH_TRU")

      int   ngroups             The number of different waste class groups to
                                which this waste class belongs

      char  *group[MAXGROUP]    The name of each waste class group to which
                                this waste class belongs.

      BOOL  perm                TRUE if this waste class was loaded from a
                                library; FALSE if the user created it during
                                the current session.

      BOOL  selected            TRUE if the user has selected this waste
                                class for use in the current session.
```

BOOL write_me	TRUE if this waste class is to be written to the Incremental Projection File (IPF).
BOOL disabled	TRUE if user has disabled this waste class.

} WC.

typedef struct _OT { Define the OT (Operation Type) object.

char *name	Name of operation type (e.g., "SIZE_REDUCTION")
BOOL is_disposal	TRUE if the operation type is a disposal; FALSE if it is a treatment.
BOOL perm	TRUE if this operation type's name was loaded from a library; FALSE if the user created it during the current session.
BOOL selected	TRUE if the user has selected this operation type for use in the current session.
BOOL write_me	TRUE if this operation type should be written to the IPF.
BOOL disabled	TRUE if the user has disabled this operation type.

} OT.

typedef struct _WG Define the WG (Waste Generator) object.

char *name	Name of waste generator (e.g., "WHC_313").
char *desc	General comment for this waste generator.
char *distr_comment [MAXCLASS]	A comment about the distribution of each waste class generated by this waste generator.
char *proj_comment [MAXCLASS]	A comment about the projected volume of each waste class generated by this waste generator.
int startyear	Year in which this waste generator starts operation.
int nyears	Number of years this waste generator operates.

BOOL class[MAXCLASS]	For each waste class, TRUE if the waste class is produced by this waste generator.
float volume [MAXCLASS;MAXYEAR]	The projected volume of each waste class generated by this waste generator by year.
float distr[MAXOP]	The distribution fraction for each waste-class-specific operation on the output of this waste generator.
int ngroups	The number of different waste generator groups to which this waste generator belongs.
char *group[MAXGROUP]	The name of each waste generator group to which this waste generator belongs.
BOOL perm	TRUE if this waste generator's information exists in a library; FALSE if the user created it during the current session.
BOOL selected	TRUE if the user has selected this waste generator for use in the current session.
char *sc_selected	Name of scenario selected for this waste generator for this case. Consists of an SC name appended to the name of this waste generator.
char *sc_parent	Scenario name of the parent of the above scenario. NULL if the above scenario is a root and, therefore, has no parent.
int parent_index	Index of this Waste Generator's parent. Use -1 if this waste generator has no parent.
int child_index	Index of this waste generator's child. Use -1 if this waste generator has no child.
BOOL write_me	TRUE if this waste generator should be written to the IPF.
BOOL disabled	TRUE if the user has disabled this waste generator.

} WG.

typedef struct _OP { Define the OP (Operation) object. (An Operation is either a treatment or a disposal. Also, "Operation" implies "waste-class-specific operation," which is a concatenation of "general-operation type" and "waste class.")

char *name	Name of waste-class-specific operation (e.g., "SIZE_REDUCTION_TRU_RH").
char *optype	Name of general treatment or disposal type (e.g., "SIZE_REDUCTION").
char *wstclass	Name of waste class this operation handles (e.g., "RH_TRU").
char *desc	General comment for this operation.
char *capac_comment	Comment describing the capacity of this operation.
char *distr_comment	Comment describing the distribution for this operation.
int startyear	Year in which this operation begins.
int nyears	Number of years this operation lasts.
float capacity[MAXYEAR]	Capacity of this operation by year.
BOOL is_disposal	TRUE if the operation is a disposal; FALSE if it is a treatment.
float distr[MAXOP]	The distribution fraction for each operation performed on the output of this treatment operation.
float ratio	The volume adjustment factor for waste going through this treatment operation.
float store_rate	Cost of storage per unit volume at this operation.
float treat_rate	Cost of this operation per unit volume.
int ngroups	The number of different treatment or disposal groups to which this operation belongs.
char *group[MAXGROUP]	The name of each treatment or disposal group to which this operation belongs.
BOOL perm	TRUE if this operation's name was loaded from a library; FALSE if the user created it during this session.
BOOL selected	TRUE if the user has selected this operation for use in the current session.

char *sc_selected	Name of scenario selected for this operation for this case. Consists of an SC name appended to the name of this operation.
char *sc_parent	Scenario name of the parent of the above scenario. NULL if the above scenario is a root and, therefore, has no parent.
int parent_index	Index of this operation's parent. Use -1 if this operation has no parent.
int child_index	Index of this operation's child. Use -1 if this operation has no child.
BOOL write_me	TRUE if this operation should be written to the IPF.
BOOL disabled	TRUE if the user has disabled this operation.

} OP.

typedef struct _CASE { Define the CASE Object. This object holds the information that is written to the header of each ".cas" file.

char name[MAXNAM]	Name of the case.
char basename[MAXNAM]	Name of the case's parent (if any).
char comment[MAXCOM]	Comment describing the case.
char author[MAXNAM]	Person who created the case.
char baseauthor[MAXNAM]	Person who created the case's parent (if any).
char datafile[MAXCOM]	Name of Reference Projection File for the case (RPF).
char sumryfile[MAXCOM]	Name of Reference Projection Summary file for the case (RPS).
char incdatafile[MAXCOM]	Name of IPF for the case.
char incsumryfile[MAXCOM]	Name of Incremental Projection Summary file for the case (IPS).
char datadict[MAXCOM]	Name of Data Dictionary File for case.

char result[MAXCOM]	Name of Result File where the case's output is written.
int firstyear	First year covered by the case.
int lastyear	Last year covered by the case.
BOOL modified	TRUE if this case has been modified in a way that is different from its parent.

} CASE.

2.2 <u>USER INTERFACE FUNCTION DESCRIPTIONS</u>

The SWPM User Interface is an OS/2 Presentation Manager (PM) application composed of numerous data structures and procedural function descriptions. The functions are implemented in Microsoft C Version 5.1 and Presentation Manager Resource Compiler (RC) languages, and are packaged in eight C-language modules, two RC modules and one Icon module.

2.2.1 <u>Main Program Module: UIMAIN.C</u>

As with all C programs, execution begins with function "main," which is the only function defined in UIMAIN.C. This function implements standard PM boilerplate to register two window classes, create two frame windows (one being a child of the other), and invoke the standard PM event loop. It also starts a timer that's used to check the status of ADL, which runs SWPM and the ORW.

2.2.2 <u>Window Procedure Module: UIWIN.C</u>

This module contains the two PM window procedures that service events generated when the user interacts with the two main windows of the application. These routines are both standard PM client window procedures; their structure is defined and described in the Microsoft OS/2 programmer's reference documentation. The menu structure for these windows is defined in the Resource Compiler source file UISWPM.RC, described later in this report.

<u>Level1WndProc</u>

This routine responds to user selections from the top-level menus, including the hierarchy under the "Setup/Run" and "Exit" action-bar items.

For most of the menu choices, this routine uses PM function WinDlgBox to display a dialog box for the user to interact with. When the user selects the "Edit" menu item, this routine activates its child window, which is serviced by Level2WndProc. In response to the "Run" and "Generate Reports" menu items, the routine uses "run_adl" to create a subprocess to run ADL to run SWPM or REPORT (the ORW). The routine also manages the enabling and disabling of menu items in its frame window to guide the user through the logical succession of steps in a SWPM session.

The important actions that this routine performs in response to each menu selection by the user are summarized below:

Menu Item	Processing by Level1WndProc
Sign On	Invoke SignonDlgProc to get and validate username/password. Quit if not valid.
Create New Case	Invoke "filename" to solicit new case name, then library name (i.e., RPF/RPS/IPF/IPS filename), then data dictionary (DD) name. Then invoke "load_summary" to fill case header and load the RPS/IPS files specified.
Use Old Case	Invoke "filename" to solicit old case name, then "readcase_header" to get the library file names. Then invoke "readcase" to read the entire case log file and preselect the objects indicated therein.
Preselect	Invoke PreselectDlgProc.
Load	Invoke "pre_load_check," then "loadpfi," then "post_load_check."
Edit	Use PM routine WinSetWindowPos to activate the child window, thereby enabling Level-2 menu functions serviced by Level2WndProc.
Run	Invoke "write_case_files" and "writesmc" to create and update the necessary files, to save the current case, and to run SWPMOD. Then invoke "run_adl" to spawn a subprocess to run ADL, which runs SWPM asynchronously with respect to the User Interface application.
Report Content	Invoke ContentDlgProc.

Menu Item	Processing by Level1WndProc
Report Order	Invoke OrderDlgProc, which uses WinDlgBox's "CreateParams" parameter to determine which of eight object types to present to the user for ordering.
Report Generate	Invoke "filename" to solicit the name of the RSL file from which the user wishes to generate reports. Then invoke "writepcf" to create a new print control file. Then invoke "run_adl" to spawn a subprocess to run ADL, which runs REPORT asynchronously with respect to the User Interface application.
Report Print	Invoke "filename" to solicit the name of the REP file for which the user wishes a hardcopy. Then invoke "do_command" to execute ORWLASER.CMD, which sends the specified file to the laser printer.
Stop Model	Invoke "kill_adl" to stop the instance of ADL that is running SWPMOD.
Stop Report	Invoke "kill_adl" to stop the instance of ADL that is running REPORT.
Status	Build a string that describes the current status of the subprocesses (if any) that are running ADL and use PM routine WinMessageBox to display it.
Exit	Give the user a chance to change his/her mind if ADL is running. Then ask the user if changes are to be saved and, if so, invoke "write_case_files." Invoke "redirect_output" to restore the standard output stream, and post a message to cause this window procedure to quit.

Level2WndProc

This routine responds to user selections from the Level-2 menus, which comprise the functions that allow the user to edit SWPMOD input data. Action-bar items serviced by this routine include "Waste Generator," "Operation," "Waste Class," "Case Description," and "Close." In response to selections from these items, the routine uses WinDlgBox or WinLoadDlg to pop up modal or modeless dialog boxes, respectively, with which the user interacts to edit SWPMOD input data. When the user selects "Close," the routine sends itself a message to deactivate the level 2 window. Level2WndProc also manages the enabling and disabling of its own menu items to guide the user through the data editing process.

The important actions that this routine performs in response to each menu selection by the user are summarized below:

Menu Item	Processing by Level2WndProc
WG Select	Invoke SelectDlgProc to solicit a waste generator and its waste class from the user.
WG Proj/Dist	Invoke "select_child_wg" to access the child (editable) version of the currently selected waste generator object, creating one if necessary. (This is done when the user is about to modify an object's data.) Then use PM routine WinLoadDlg to invoke modeless dialog procedure WGPDDlgProc, which interacts with the user to edit projection and distribution data.
WG General Comment	Invoke "select_child_wg," then invoke CommentDlgProc. (CommentDlgProc is general-purpose; the specific type of comment being edited is passed to it in WinDlgBox's "CreateParams" parameter.)
WG Dist. Comment	Invoke "select_child_wg," then CommentDlgProc.
WG Proj. Comment	Invoke "select_child_wg," then CommentDlgProc.
WG Groupings	Invoke "select_child_wg," then GroupDlgProc. (GroupDlgProc, like CommentDlgProc, is general-purpose.)
OP Select	Invoke SelectDlgProc to solicit an operation from the user.
OP Capacity/Dist	Invoke "select_child_op" to access the child (editable) version of the currently selected Operation object, creating one if necessary. (This is done when the user is about to modify an object's data.) Then use PM routine WinLoadDlg to invoke modeless dialog procedure OPCDDlgProc, which interacts with the user to edit capacity and, in the case of Treatment Operations, distribution data.

Menu Item	Processing by Level2WndProc
OP General Comment	Invoke "select_child_op," then invoke CommentDlgProc. (CommentDlgProc is general-purpose; the specific type of comment being edited is passed to it in WinDlgBox's "CreateParams" parameter.)
OP Dist. Comment	Invoke "select_child_op," then CommentDlgProc.
OP Capac. Comment	Invoke "select_child_op," then CommentDlgProc.
OP Groupings	Invoke "select_child_op," then GroupDlgProc. (GroupDlgProc, like CommentDlgProc, is general-purpose.)
WG Select	Invoke SelectDlgProc to solicit a waste class from the user.
WC Groupings	Invoke GroupDlgProc.
Describe Case	If the user has modified the case and has not provided a new name yet, invoke "filename" to solicit a new case name. Then invoke CaseDlgProc to solicit case info, and then "writecase" to create a new case log (.CAS) file.
Close	Send this window a message telling it to close itself.

2.2.3 Dialog Procedure Module: UIDLG.C

This module contains all of the PM dialog procedures that produce dialog windows as child windows of level 1 or level 2 windows described above, and service events generated when the user interacts with these windows. It is in response to events from dialog windows that the SWPM User Interface performs most of its activity; therefore, the routines defined in this module are responsible for most of the work done by the User Interface application. Each of the routines contained in this module are standard PM dialog procedures; their structure is defined and described in the Microsoft OS/2 programmer's documentation. The dialog windows that these procedures support are defined in the module UIDLG.DLG, which is created and maintained interactively with the Microsoft OS/2 Presentation Manager Dialog Box Editor.

<u>ContentDlgProc</u>

This dialog procedure displays the contents of the file SWPM.PCD in a list box (see Chapter 6.0, <u>SWPM FILES</u>) and allows the user to pick one or more output report tables from the list. The choices are stored internally and will be used to create a new version of SWPM.PCF when the user selects the "Generate Reports" function.

<u>OrderDlgProc</u>

This dialog procedure displays two list boxes. The left-hand list box contains a list of all object classes that can be printed by the ORW; the right-hand list box is initially empty. The user is allowed to select items from the left-hand list box. As the items are selected, they are displayed on the right in the order selected. Selecting items from the right-hand list box deletes them from that box. By so manipulating the contents of the two list boxes, the user can build up in the right-hand list box a list of objects in any desired order. This order is stored internally for later output to SWPM.PCF for use by the ORW.

<u>ScenarioDlgProc</u>

This dialog procedure is used during the preselection process (when the user picks the "Preselect" menu item) as well as during the data editing process. It displays a list box and a related entry field, and allows the user to specify a general scenario name by selecting from the list box or typing into the entry field. The user is also allowed to delete a scenario name under certain conditions.

<u>NewWGDlgProc</u>

This dialog procedure allows the user to create a new waste generator object. It displays an entry field for the user to enter a new waste generator name, and a list box filled with waste classes. From the list box, the user specifies the waste classes that the new waste generator produces.

<u>NewOPDlgProc</u>

This dialog procedure allows the user to create a new treatment or disposal operation object. It displays two list boxes, one filled with names of existing operation Types and the other filled with waste classes. An entry field is also provided for the user to enter the name of an entirely new operation type. The user can create a new operation by picking an existing operation type from the left-hand list box and matching it with a waste class from the right-hand list box, or a new operation type can be entered. Radio buttons are provided to indicate whether the new operation type is a treatment or a disposal. This new operation type can then be matched with a waste class.

<u>CaseDlgProc</u>

This dialog procedure displays comments about the current case's parent case (if any) and allows the user to enter a new comment about the current case. The period of time covered by the case can also be specified.

<u>PasswordDlgProc</u>

This dialog procedure, invoked from SignonDlgProc, displays a prompt for the user to enter his/her password.

<u>SignonDlgProc</u>

This dialog procedure queries the user for the username, then invokes PasswordDlgProc to obtain the user's password, which it validates against the file SWPM.UPF. If SWPM.UPF indicates that the user is privileged, this fact is noted for subsequent use.

<u>PreselectDlgProc</u>

This dialog procedure allows the user to preselect, from three dialog boxes, the waste classes, waste generators and operations that are to be loaded from the RPF/IPF for subsequent editing and/or processing by SWPM. The list boxes are filled from the contents of the currently-specified summary files (RPS/IPS). This procedure invokes ScenarioDlgProc to solicit waste generator and operation scenario names from which to look up the data in the RPF/IPF at load time. Push buttons allow the user to select/deselect all items in any of the three list boxes.

SelectDlgProc

This dialog procedure is used during editing of data for waste generators, operations and waste classes. Its purpose is to allow the user to select the specific object to edit. It is a general-purpose routine, and one of its parameters specifies what type of object is to be selected. Depending on the object type, the routine displays one or two list boxes; two are displayed for waste generators since the user must not only pick a waste generator but also one of the waste classes it produces. When selecting waste generators or operations, the user is also given the option, via a push button, of creating a new object of the corresponding type. In these cases, this routine invokes NewWGDlgProc or NewOPDlgProc, as appropriate.

CommentDlgProc

This dialog procedure is used during the editing process to display the comment history of SWPM data objects and to allow the user to enter new comments. This single, general-purpose procedure handles the following types of comments:

- Waste Generator (General)
- Waste Generator (Projection)
- Waste Generator (Distribution)
- Treatment & Disposal Operation (General)
- Treatment & Disposal Operation (Capacity)
- Treatment Operation (Distribution)
- Waste Class (General).

As with other general-purpose dialog procedures, one of this routine's parameters, which is passed to it via WinDlgBox's "CreateParams" parameter, specifies which of these types of comments should be displayed and solicited.

GroupDlgProc

This dialog procedure (another general-purpose dialog procedure) uses two list boxes, an entry field, and several push buttons to allow the user to create and rearrange groups of SWPM objects. This procedure can handle groups of waste generators, treatment and disposal operations, and waste classes, as well as groups that combine these objects.

WGPDDlgProc

This modeless dialog procedure implements the main editing window for waste generator data. It is implemented as a modeless dialog to permit the user to view and enter comments without dismissing it. It presents the user with a list box and an entry field for viewing and editing projection data by year. Two list boxes and an entry field are provided for viewing and editing distribution data. A push button is provided to allow the user to create a new treatment or disposal operation without dismissing this window.

OPCDDlgProc

This modeless dialog procedure implements the main editing window for treatment and disposal operation data. It is implemented as a modeless dialog to permit the user to view and enter comments without dismissing it. It presents the user with a list box and an entry field for viewing and editing treatment and disposal capacity data by year. Two list boxes and an entry field are provided for viewing and editing treatment distribution data; these are disabled when a disposal is being edited. Three entry fields allow editing of storage cost, treatment/disposal cost, and (for treatments only) throughput volume ratio. A pushbutton is provided to allow the user to create a new treatment or disposal operation without dismissing this window.

FilenameDlgProc

This dialog procedure, used to support the "filename" procedure in module UIFILEIO.C, uses a list box and an entry field to prompt the user for an existing or new file name with a given filetype.

2.2.4 Process Control Module: UISPAWN.C

The procedures in this module create and manage subprocesses that run ADL programs asynchronous to the SWPM User Interface application.

REDIRECT_OUTPUT

This routine redirects standard output to the specified file, or to the actual standard output stream if the specified file is NULL.

RUN_ADL

This routine creates a subprocess that runs ADL with the specified ADL source file, which in turn will read the specified Model Control File.

ADL_STATUS

This routine obtains the current status of the subprocesses that run ADL with SWPM and REPORT and returns this information to its caller.

KILL_ADL

This routine uses DosKillProcess to abolish the subprocess that is running ADL with the specified program, either SWPMOD or REPORT.

2.2.5 File Input/Output Module: UIFILEIO.C

This module contains procedures that handle file input and output for the User Interface. For a description of the files it handles (see Chapter 6.0, SWPM FILES).

LOAD_SUMMARY

This routine reads the RPS (.RPS) and IPS (.IPS) files and creates SWPM data objects (waste classes, operation types, waste generators, operations and scenario names) from the summary information contained in these files.

LOADPFI

This routine loads information into the SWPM data objects created by LOAD_SUMMARY. LOAD_PFI is activated by selection of the Load function from the Setup/Run menu. It calls PFI routines to accomplish this. After information is loaded, LOADPFI builds a list of group names from the grouping information in the waste generator and operation data objects.

WRITEPCF

This routine creates a printer control file, "SWPM.PCF."

WRITESMC

This routine Creates a Model Control (.SMC) file.

READCASE_HEADER

This routine reads the header of a Case description save (.CAS) file into the case header data structure.

READCASE

This routine reads a .CAS file header, then reads the body of the .CAS file and creates the SWPM data structures. While doing this, the routine checks for inconsistencies between the data being read from the .CAS file and the .RPS and .IPS files read previously during the session. As each inconsistency is discovered, the user is given a chance to fix it or abort the reading process. (These inconsistencies will be created if .RPS and/or .IPS files are modified independently of the User Interface.)

WRITECASE

This routine creates a new .CAS file.

WRITEPFI

This routine calls PFI_WRITE to write SWPM data objects flagged "write_me" to the IPF.

WRITE_CASE_FILES

This routine calls WRITECASE and WRITEPFI, effectively saving the case and preparing the IPF for a SWPM run. If the user has not yet given the current case a name, a name must be supplied at this time. If the case has not yet been described, this routine invokes CaseDlgProc to force the user to supply a description.

2.2.6 Consistency Check Module: UICHECKS.C

The procedures in this module check SWPM input data for consistency before and after data loading, and also check the syntax of the user's entries.

SYNTAX

This routine ensures that the string provided adheres to the rules for the name of a SWPM data object: it must be less than 32 characters in length; it must begin with a letter; and it must be composed of letters, numbers and underscores (no spaces).

PRE_LOAD_CHECK

This routine, invoked prior to loading information from the RPF/IPF, ensures that the user has preselected at least one waste generator, waste class, and operation.

POST_LOAD_CHECK

This routine, invoked after loading information from the RPF/IPF, performs a consistency check on the loaded data. It checks each waste generator preselected by the user to ensure that at least one of its waste classes was preselected. For each waste generator that fails the test, the user can eliminate the waste generator from the case or abort and start over.

2.2.7 Data Manipulation Module: UIDATMAN.C

The procedures in this module are called by higher-level procedures to manipulate data stored in SWPM data structures.

CLEARALL

This routine initializes all SWPM User Interface data structures.

MAKE_SC

This routine creates from the supplied string an instance of a "scenario" data object, which is a general scenario name that can be appended to a waste generator or and operation name to form a specific scenario name.

DESTROY_SC

This routine destroys an instance of a scenario data object. ("Destroying" a SWPM data object means freeing its storage and nulling any pointers to it in other SWPM data objects.)

MAKE_WC

This routine creates an instance of a waste class data object; the supplied string becomes the object's name.

MAKE_WG

This routine creates an instance of a waste generator data object; the supplied string becomes the object's name.

DESTROY_WG

This routine destroys an instance of a waste generator data object.

MAKE_OT

This routine creates an instance of an operation type data object from the supplied string and the boolean "Treatment/Disposal" indicator.

DESTROY_OT

This routine destroys an instance of an operation type data object.

MAKE_OP

This routine creates an instance of an operation data object from the operation type and waste class supplied as indices to preexisting component data objects.

DESTROY_OP

This routine destroys the instance of an operation data object specified by indices to its components.

SELECT_CHILD_WG

This routine opens the current waste generator data object, as identified by a global variable set when the user selects a waste generator, for editing by the user. "Opening" means accessing the "child" version of a Waste Generator instance, creating a new one if necessary.

SELECT_CHILD_OP

This routine opens the current operation data object, as identified by a global variable set when the user selects an operation, for editing by the user. "Opening" means accessing the "child" version of an operation instance, and creating a new one, if necessary.

SAVE_WG_VALUES

This routine saves the contents of the specified waste generator data object in a global waste generator save area.

RESTORE_WG_VALUES

This routine restores the contents of the global waste generator save area to the specified waste generator data object. (The save/restore routines... this one and the next three to be described, are used during the editing process to facilitate a "Cancel" function.)

SAVE_OP_VALUES

This routine saves the contents of the specified operation data object in a global operation save area.

RESTORE_OP_VALUES

This routine restores the contents of the global operation save area to the specified operation data object.

LOOKUP

This routine returns to its caller the index of the specified SWPM data object of the specified type.

UPDATE_WCLIST

This routine updates the waste class list box in the "Preselect" dialog box with the current waste class names and selections.

UPDATE_WGLIST

This routine updates the waste generator list box in the "Preselect" dialog box with the current waste generator names and selections.

UPDATE_OPLIST

This routine updates the operations list box in the "Preselect" dialog box with the current operation names and selections.

2.2.8 Utility Procedures Module: UIUTIL.C

This module contains low-level utility routines called by higher-level routines in the SWPM User Interface application.

MAKETITLEBAR

This routine constructs a string for use in the title bar of the top-level window, from the SWPM User Interface version identifier and the current case name.

ENABLEMENU

This routine enables the specified item in the specified menu action bar.

DISABLEMENU

This routine disables the specified item in the specified menu action bar.

UPPER

This routine converts the supplied string to upper case.

DO_COMMAND

This routine performs the OS/2 command contained in the supplied string.

DTB

This routine removes trailing spaces and control characters from the supplied string.

ALLOCSTR

This routine allocates dynamic storage for the supplied string, copies the string into it, and returns a pointer to the new string.

GoodBeep

This routine causes the computer to emit a tone sequence implying "success."

BadBeep

This routine causes the computer to emit a tone sequence implying "failure."

CareBeep

This routine causes the computer to emit a tone sequence implying "be careful."

NOTE

This routine invokes "BadBeep" and displays the supplied string in a message box and waits for the user's acknowledgment.

TIMEDATE

This routine returns the current time and date as a string.

2.2.9 Menu Resource Module: UISWPM.RC

This Microsoft OS/2 Presentation Manager Resource Compiler source file defines the menu structure for both of the SWPM User Interface windows. The contents of this file are given below:

```
/* uiswpm.rc */
/* ========= */

/* ==================================================================*/
/* Resource source file for the SWPM User Interface Application...   */
/* Defines the menu structure for the Level 1 and 2 windows...     */
/* Includes resource source file created by Dialog Box Editor     */
/* ================================================================== */

#include <os2.h>
#include "uiswpm.h"
rcinclude uidlg.dlg

POINTER        ID_RESOURCE_1000 uiswpm.ico

MENU                                    ID_RESOURCE_1000
    {
    SUBMENU "Setup/Run,"                    IDM_SETUP_RUN
        {
        SUBMENU "Initialize,"                   IDM_INITIALIZE
        {
            MENUITEM "Sign On,"                     IDM_SIGNON
            MENUITEM "Create New Case,"             IDM_NEWCASE,,MIA_DISABLED
            MENUITEM "Use Old Case,"                IDM_OLDCASE,,MIA_DISABLED
        }
        MENUITEM "Preselect,"                   IDM_PRESELECT,,MIA_DISABLED
        MENUITEM "Load,"                        IDM_LOAD,,MIA_DISABLED
        MENUITEM "Edit,"                        IDM_EDIT,,MIA_DISABLED
        MENUITEM "Run,"                         IDM_RUN,,MIA_DISABLED
        SUBMENU  "Report,"                      IDM_REPORT
        {
            MENUITEM "Content,"                 IDM_CONTENT
            SUBMENU  "Order,"                       IDM_ORDER
            {
                MENUITEM "Waste Generators,"
IDM_ORDER_WG,,MIA_DISABLED
                MENUITEM "Waste Classes,"
IDM_ORDER_WC,,MIA_DISABLED
                MENUITEM "Treatments,"
IDM_ORDER_TRT,,MIA_DISABLED
                MENUITEM "Disposals,"
IDM_ORDER_DP,,MIA_DISABLED
                MENUITEM "Waste Generator Groups,"
IDM_ORDER_GG,,MIA_DISABLED
                MENUITEM "Waste Class Groups,"
IDM_ORDER_GC,,MIA_DISABLED
                MENUITEM "Treatment Groups,"
IDM_ORDER_GT,,MIA_DISABLED
                MENUITEM "Disposal Groups,"
IDM_ORDER_GD,,MIA_DISABLED
            }
            MENUITEM "Generate,"                    IDM_GENERATE,,MIA_DISABLED
            MENUITEM "Print,"                       IDM_PRINT,,MIA_DISABLED
        }
        SUBMENU  "Stop,"                        IDM_STOP
        {
                MENUITEM "Stop Model,"              IDM_STOPMOD,,MIA_DISABLED
                MENUITEM "Stop Report Generator,"   IDM_STOPREP,,MIA_DISABLED
        }
        MENUITEM "Status,"                      IDM_STATUS
        }
    MENUITEM "Exit,"                        IDM_EXIT
    }

POINTER        ID_RESOURCE_2000 uiswpm.ico

MENU                                    ID_RESOURCE_2000
    {
    SUBMENU "Waste Generator,"              IDM_WG
        {
        MENUITEM "Select,"                      IDM_WG_SEL
        MENUITEM "Projection/Distribution,"     IDM_WG_PD,,MIA_DISABLED
        MENUITEM "General Comment,"              IDM_WG_GCOM,,MIA_DISABLED
        MENUITEM "Distribution Comment,"         IDM_WG_DCOM,,MIA_DISABLED
        MENUITEM "Projection Comment,"           IDM_WG_PCOM,,MIA_DISABLED
        MENUITEM "Groupings,"                    IDM_WG_GROUP,,MIA_DISABLED
        }
    SUBMENU "Operation,"                     IDM_OP
        {
        MENUITEM "Select,"                       IDM_OP_SEL
        MENUITEM "Capacity/Distribution/Cost/Ratio,"IDM_OP_CD,,MIA_DISABLED
        MENUITEM "General Comment,"              IDM_OP_GCOM,,MIA_DISABLED
        MENUITEM "Distribution Comment,"         IDM_OP_DCOM,,MIA_DISABLED
        MENUITEM "Capacity Comment,"             IDM_OP_CCOM,,MIA_DISABLED
        MENUITEM "Groupings,"                    IDM_OP_GROUP,,MIA_DISABLED
        }
    SUBMENU "Waste Class,"                   IDM_WC
        {
        MENUITEM "Select,"                       IDM_WC_SEL
        MENUITEM "Groupings,"                    IDM_WC_GROUP,,MIA_DISABLED
        }
    MENUITEM "Case Description,"             IDM_CASE
    MENUITEM "Close,"                        IDM_CLOSE
    }
```

2.2.10 Dialog Box Resource Module: UIDLG.DLG

This Microsoft OS/2 Presentation Manager Resource Compiler source file defines all of the SWPM User Interface dialog boxes. It is created and maintained by the Microsoft OS/2 Presentation Manager Dialog Box Editor, which uses the corresponding binary and header files UIDLG.RES and UIDLG.H. None of these files should be touched by anything other than the Dialog Box Editor; to violate this rule is to court disaster.

2.2.11 Icon Resource Module: UISWPM.ICO

This module is a binary file that defines the 32-by-32 pixel PM icon for the SWPM system. It is created and maintained by the Microsoft OS/2 Presentation Manager Icon Editor.

3.0 PROJECTION FILE INTERFACE

This chapter describes the projection file interface (PFI), which provides a link between the User Interface and the data library. The library includes several data files labeled .RPF, .RPS, .IPF, and .IPS. The PFI is written in C and is used as subroutine calls from the User Interface. The User Interface calls the PFI to load data from the .RPF and .IPF files into the User Interface data structures. When the user has finished editing the data, the User Interface calls the PFI to update the incremental projection and summary files.

3.1 PFI OVERVIEW

The PFI is effectively divided into two pieces, the portion that loads data into the User Interface structures and the portion that writes the data from the User Interface structures into the incremental data library.

The PFI uses the SCANNER and LOADDATA modules that were written as part of the ADL engine. In the load phase, it calls the LOADDATA module and "intercepts" the values that are normally stored in ADL data structures and stores them in the User Interface data structures instead. During the write phase, the PFI uses the SCANNER to read the existing incremental files and echoes the data through to the updated incremental files along with the extra information contained in the User Interface data structures.

3.1.1 Processing Narrative

The general flow of the load phase is:

- Create the list of macros (of the form "NAME=NAME") using the given list and the summary files. These macros are given to the SCANNER so that it will read only the desired data from the projection files. A description of the #IFDEF structure used in the data files is described in the SWPM FILES section of this document.
- Call the LOADDATA module.
- Whenever LOADDATA finds a data value, it calls PFI_STORE_VAL, which loads the data in the User Interface data structures.
- Free up necessary memory.

The flow of the write phase is:

- Get list of new objects to declare (for the header section of the data file).
- Update the IPS.
- If the IPF already exists, get the list of objects (waste classes, waste generators, and operations) from its header section.
- Merge the list of objects from the existing incremental file with the new objects in the User Interface data structures.
- Write the merged lists of objects to the header section of the new incremental file.
- If the old incremental file exists, copy its declaration section to the new incremental file.
- Add the necessary new declarations to the new incremental file.
- If the old incremental file exists, copy its data section to the new incremental file.
- Add the necessary new data statements to the new incremental file.
- Free up necessary memory.

3.1.2 Data Flow

When the User Interface loads data, PFI_LOAD_PROJECTION is called, and the PFI reads the RPS and RPF files and the IPF and IPS files if they exist. The User Interface then loads the data sent to it from LOADDATA into the User Interface data structures.

When the user has finished editing data, the User Interface calls PFI_WRITE and the PFI updates the IPF and IPS files. If those files do not exist, they are created. Figure 3.1 illustrates the data flow involved.

3.2 FUNCTION DESCRIPTIONS

Because the PFI uses the LOADDATA module from ADL (and, thus, the SCANNER, BUFFER, SYMTBL, and some PARSER routines), the PFI module is under the configuration management of the ADL code. Also, it uses several of the "include" (*.h) files from the ADL code but has its own specific "include"

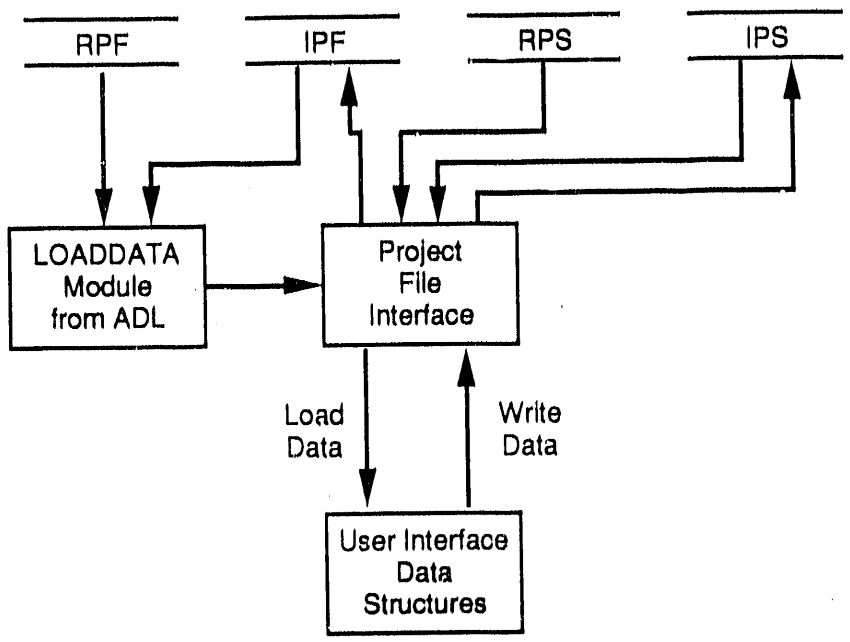


FIGURE 3.1. Data Flow

file (PFI.H) which defines some internal data structures. Each of the routines in the PFI is in a separate file and has associated unit tests according to ADL Configuration Management. The routine names start with "PFI_" followed by a name describing the routine. The following sections give a description of the primary routines in the PFI module. Some of the auxiliary routines that perform tasks such as freeing memory and parsing sections of the summary files are not described.

3.2.1 Projection File Loading

The portion of the PFI that loads the data files is composed of only a few PFI routines. The LOADDATA modules from ADL do the bulk of the work.

PFI_LOAD_PROJECTION

This routine is called to initiate the loading of a projection file. It initializes the SCANNER module for LOADDATA, loads the :TREE section from the given summary file, and calls LOADDATA to process the projection file. PFI_LOAD_PROJECTION is called once for each projection file; these currently include the RPF and the IPF.

PFI_GET_MACROS

This routine reads the summary file, which describes the projection file, and creates the list of macros from the :TREE section in the summary file and the scenario list. The macros list is used to initialize the SCANNER so that it will read only the required data from the projection file. See the section in this document that describes the file formats. This routine puts the scenario list, which was passed in to PFI_LOAD_PROJECTION, in the macros list and adds all the scenarios' "ancestors" read from the summary file(s).

PFI_STORE_VAL

PFI_STORE_VAL is the routine that "intercepts" the data calls out of the LOADDATA module from the LDU_SENDDATA routine. Each piece of data sent from the LOADDATA module includes the frame name, slot name, data type, and data value. PFI_STORE_VAL interprets the name of the frame to determine if the data is for a waste generator or an operation. The correct User Interface data structure is then located, and the value is stored.

3.2.2 Projection File Writing

The User Interface allows the user to modify data as well as view it, and once user has completed the process, the User Interface calls PFI_WRITE to save the updated values in the IPF and the IPS.

When the PFI updates the IPS file, all it has to do is open the file and append and write the necessary data. Updating the IPF file is more difficult. The PFI must add the new declarations to the existing HEADER section of the IPF file, copy the existing DATA section, and then append the new data to the bottom of the file. These actions are accomplished by reading the old IPF file, writing to a temporary file, and then renaming the temporary file when the operation is done. The majority of the PFI routines are devoted to performing this seemingly simple task.

PFI_WRITE

This routine is the driver for updating the IPF and IPS files. It writes some of the headers and calls the necessary routines.

PFI_WRITE is application specific, and it creates the projection file in a very strict format so that it knows the order of the declaration section when items must be appended to it.

<u>PFI_WRITE_SUMMARY</u>

This routine updates the incremental summary file. It traverses the incremental information list and writes pertinent information to the file.

<u>PFI_ADD_DECL</u>

This routine adds all new object information to the declaration lists. These lists are used by PFI_WRITE to create the header section of the IPF. Every piece of data that the user has added or modified is put into the IPF as a data statement. For the data statement to be valid, the header section must list the object and its indices (if it is an array). This routine saves all the array index information.

<u>PFI_GET_LISTS</u>

PFI_GET_LISTS reads the first part of the header section of the existing IPF and saves the information in the lists and ranges that are used in the object declarations.

<u>PFI_MERGE</u>

PFI_MERGE adds the nonredundant values (from User Interface data structures) to the lists containing the information from the existing projection file.

<u>PFI_WRITE_LIST</u>

PFI_WRITE_LIST writes the contents of a list to a file. It is a general purpose routine that does some formatting of the list. It is used for writing the lists of waste generators, operations, and waste classes in the declaration (HEADER) section of the projection file.

<u>PFI_COPY_DECL</u>

This routine copies the declarations from the existing projection file to the new file.

PFI_NEW_DECL

PFI_NEW_DECL creates declarations for all variables in the declaration list. These are the objects that were added by the user through the User Interface and were not already in the existing projection file.

PFI_GET_INDICES

This routine takes an object as an argument and determines the index names associated with it. These index names are used in both the header and data section of the projection file. For example, the WG_DIST_F array is dimensioned by waste class and operation so PFI_GET_INDICES returns pointers to the strings "WC" and "OPERATION" if it gets "WG_DIST_F" as an argument.

PFI_COPY_DATA

PFI_COPY_DATA copies the existing IPF's data section to the new projection file.

PFI_NEW_DATA

PFI_NEW_DATA adds the new portion of the data section to the new projection file, and includes any data that the user modifies or adds through the User Interface.

4.0 SOLID WASTE PROJECTION MODEL

The SWPM is the code that calculates the volumes of wastes through operations (treatments and disposals) based on data from the Reference Projection Libraries and the Incremental Projection Libraries. These libraries contain projected volumes of wastes from waste generators by year, distribution fractions by waste classes to operations, volume adjustment factors for operations, and distribution fractions from operations to other operations, and other relevant information.

The SWPM produces a "result" file containing the results of the model's calculations as well as other information carried through from the projection libraries. The result file is used by the ORW to generate desired reports.

The SWPM is written in ADL, a proprietary programming language developed at PNL specifically for modeling physical systems. ADL can be run from the OS/2 prompt, but is generally expected to be run from the SWPM User Interface.

4.1 SWPM OVERVIEW

The general idea of the model is to read a control file that defines what waste generators and operations are included in a case and create "instances" of those waste generators and operations. Waste generators are instances of the WASTE_GEN class, and operations (treatments and disposals) are instances of the OPERATION class. Data (volumes, capacities, etc.) are then read into the objects, each of which represents a treatment or disposal. Finally, all values are calculated over the given years. After the calculations are done, the values are printed to the result file along with some information that is echoed through from the input data files, (e.g., comments).

4.1.1 Processing Narrative

The general flow of the model is as follows. At the prompt for the SWPM Model Control file (SMC):

- Load the model classes.
- Process the SMC and dynamically create the necessary objects.

- After loading the projection libraries, enter the sources and sinks for each operation class.
- Fill the INVENTORY, PROCESSED, TREAT_COST, STOR_COST, and AMNT_RECVD arrays by querying each of the operations at each year.
- Print out the result file.

4.1.2 Data Flow

The objects in the OPERATION class are the key to the model. Each one represents a disposal or treatment, and is "smart enough" to know how to calculate all the necessary volumes for a year. Objects assume, though, that all the volumes for the previous year were already calculated. Members of the WASTE_GEN class have very few associated methods and are used mainly for storing data from the projection libraries. Figures 4.1 and 4.2 represent an object from the WASTE_GEN and OPERATION classes, respectively.

Figure 4.3 represents the model if it were given three waste generators (PNL, PUREX, and HWVP), which are objects of the WASTE_GEN class; four treatment operations (COMP, INCIN, SEG, and VR); and two disposal operations (DISP1 and DISP2), which are objects of the OPERATION class. The solid lines indicate subclass relationships, and the textured lines indicate member relationships (instances).

4.2 FUNCTION DESCRIPTIONS

In the SWPM, the functions are grouped together in classes (frames) except for the main program and some general purpose routines. Each class is defined in a separate file, which has a .h extension and a name corresponding to the frame name. Version 1.1 of OS/2 allows file names to have only

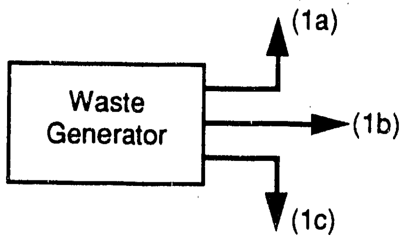
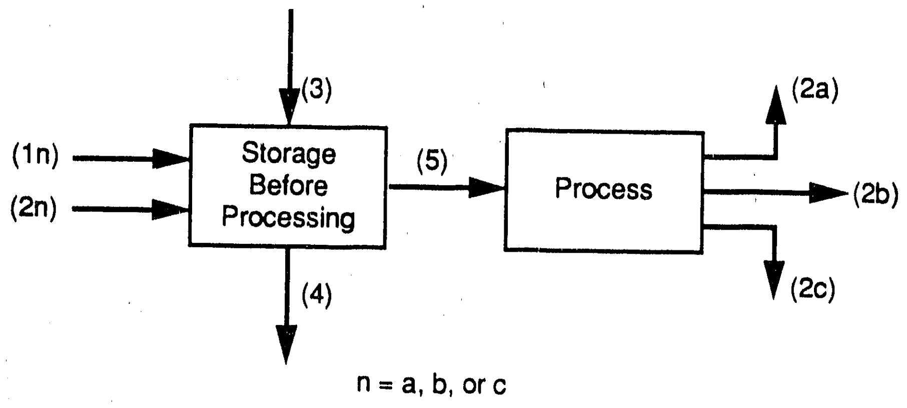


FIGURE 4.1. Waste Generator



(1n) - The output from a waste generator, which is also the input into storage before process.

(2n) - The output from a treatment process, which is distributed into storage before the next process.

(3) - The volume coming into storage from the previous time unit.

(4) - The volume going into storage for the next time unit. This is the year end inventory in the storage for this operation.

(5) - The volume going through the process. This has a capacity limit associated with it.

FIGURE 4.2. Treatment Operation

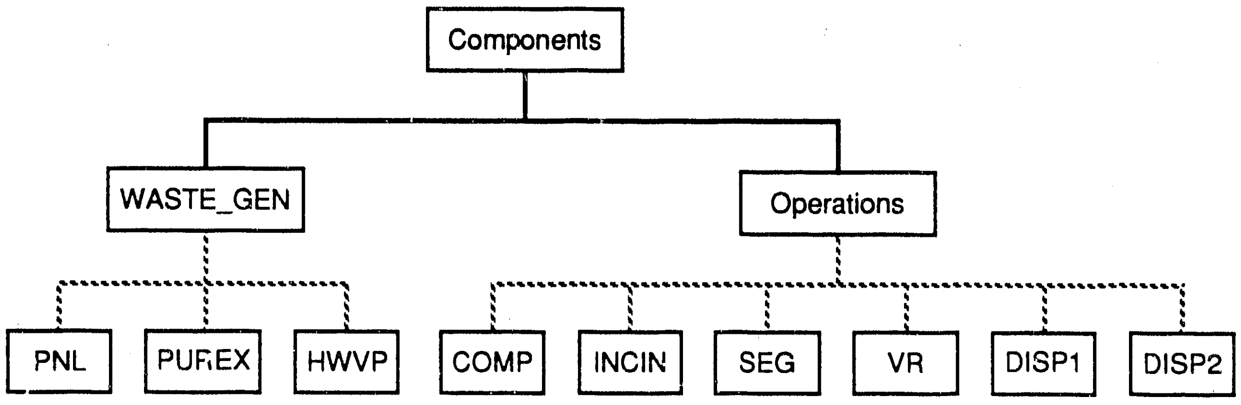


FIGURE 4.3. Object Class Hierarchy

8 characters; thus, the file names are not always the same as the frame names. Variables specific to each frame are declared at the top of each file and have descriptions with them (see the following listing).

FRAME NAME	PC FILE NAME	VAX FILE NAME
Main Program	SWPM.ADL	SWPM.ADL
General Routines	METHOD.H	METHOD.h
COMPONENTS	CMPNENTS.H	COMPONENTS.h
SMC	SMC.H	SMC.H
MODELS	MODELS.H	MODELS.H
REPORT_WRITER	REPWRIT.H	REPORT_WRITER.H

4.2.1 Main Program

The file SWPM.ADL contains the main program that drives the model. The actions described below occur when SWPM.ADL is run.

- Prompt for SMC fileload.
- The MODELS, COMPONENTS, SMC, and REPORT_WRITERS classes/files.
- Call the LOAD_SMCF in the MODEL_SMC class.
- Invoke the INIT_MODEL method.
- Exit if errors have occurred.
- Invoke the FILL_ARRAYS method.
- Invoke the REPORT_WRITER method.

4.2.2 General Methods

Several general purpose routines are included in the file METHOD.H. Some of these are attached to slots within operation and waste generator instances, and some are included as functions within those instances.

COMMENT_PUTDEM

This method is attached as a put-demon to all the nonarray comment objects. It appends all comments assigned to a particular variable, thus keeping the full history.

ARR_COMMENT_PUTDEM

This is the same as COMMENT_PUTDEM, but is attached to array comment objects.

UPCASE_PUTDEM

This put-demon is attached to string objects that require having strings in upper case letters.

ACTIVATE_OP_GETDEM

This method is attached to the TRT_PROCESSED slot in operation instances. When the value of TRT_PROCESSED is requested, this demon checks to see if the operation has been activated (values calculated) for the current year, and if not, it activates the processing of the operation. ACTIVATE_OP_GETDEM also checks for recursion through operations and prints the traceback path if it is attempting to recurse. Because ACTIVATE_OP_GETDEM is attached only to one slot (for the purpose of saving memory), that slot must be queried before any others, such as TRT_AMNT_RECVD, TRT_STOR_COST, TRT_TREAT_COST, etc.

RESET

The RESET method is used to reset an operation to prepare it for the current year processing by setting appropriate operation variables to ZERO.

PROCESS_OPERATION

PROCESS_OPERATION is the primary calculational method used in the SWPM. It is placed as a function in each operation instance and calculates the values for the variables in that instance. Before this method is invoked for any operation, the RESET method must be called for each of the operations.

PRINT_LIST

PRINT_LIST prints an ADL list to a given logical unit.

IS_INT

IS_INT checks a string to see if it contains only digits (i.e. if it is a valid integer).

4.2.3 COMPONENTS File

The COMPONENTS file is the super-class of WASTE_GEN and OPERATION. It has some function slots that are inherited by the subclasses (see Figure 4.3).

NEWMEMBER and NEWSUBCLASS

These methods create a member and a subclass of the owning (COMPONENTS) class, respectively. They check for the existence of the new frame before making it a member or subclass. If the frame already exists, a warning message is printed; this is against the possibility of some operation or waste generator instance having the same name as a variable or existing frame in the SWPM code.

FILL_ARRAYS

The FILL_ARRAYS method in the components class fills all the global volume/cost arrays by querying each of the operations during each year. It is an own slot so that it is not inherited by the subclasses and members of the COMPONENTS class.

4.3 MODELS

The MODELS class contains the code used to construct the model that is described in the data flow section.

MK_COMPONENTS

This method in the MODELS class creates the components (the OPERATION and WASTE_GEN frames) for the model. It also puts the PROCESS_OPERATION method in the OPERATION frame as a member slot so that it will be inherited by all operation instances.

MK_MODEL Method

MK_MODEL creates the instances of the components in the model and is invoked after the data dictionary is loaded. It also creates the waste generator slots in each frame that are dynamically dimensioned by lists and attaches appropriate demons to the slots.

INIT_MODEL Method

The INIT_MODEL method uses the data from the projection files to initialize all necessary slots in the WASTE_GEN and COMPONENT operations instances. This involves "telling" each of the instances where it receives and sends data. It also checks for distribution fractions that do not sum to 1 and makes sure that waste generators do not send waste from the wrong waste class to an operation.

PRINT_FLOW

The PRINT_FLOW routine lists each operation with the operations and waste generators that supply it and the operations that it supplies.

POST_DATADICT

After the SMC class loads the data dictionary, this routine is invoked and attaches the UPCASE_PUTDEM to necessary objects.

4.3.1 SMC (SWP Model Control)

The SMC class is used to read and process the SMC file. The SMC class includes a method used to parse each of the record types in the SMC file and a routine to process each of the records. For example the DATAFILE card has the corresponding routines PARSE_DATAFILE_CARD and PROCESS_DATAFILE_CARD. For a full description of the cards, see the SWPM Requirements Specification Document. The SMC class invokes the necessary methods to build the model and makes the calls to load the data. The slots in this class are all "own" slots because this class does not have any members or subclasses.

The general flow in the SMC class is as follows:

- Parse all of the cards.
- Process waste generator cards. This involves concatenating all the waste generators into the global list ALL_WG and storing the waste classes for each waste generator in a list. (The lists of waste categories become the WG_WC slot in each waste generator instance, and the WG_VOL array is declared using that list.)

- Process the TREATMENT cards. This entails concatenating all the operations into the global list TREATMENT.
- Create the global list COMP_LIST, which is a concatenation of ALL_WG, TREATMENT, and CATEGORY.
- Process the YEARS card. This involves creating the global range YEARS.
- Process the SUMRYFILE cards.
- Process the PROJECTIONS cards. This entails concatenating all the projection names as macros (name=name) in the list MODEL:MACROS.
- Invoke the MK_COMPONENTS method.
- Process the DATADICT cards.
- Invoke the MK_MODEL method.
- Process the DATAFILE cards.
- Check card counts (i.e., make sure enough cards were provided).

The following methods are included in the SMC class.

<u>LOAD_SMCF</u>

This is the controlling routine for the SMC class. It makes all the necessary calls to load the model control file.

<u>CHECK_CARD_COUNTS</u>

Check to make sure enough cards of each type were specified.

<u>GET_CARDS</u>

Read all the cards from the SMC file and store them in the appropriate arrays.

<u>ADD_CARD</u>

Add a card to the CARDS array.

<u>PARSE_CARD</u>

This routine checks the name of a card and calls the appropriate parse card routine.

PARSE_name_CARD

Parse each "name" card, where name is one of the card types. Each of these routines stores the necessary data from the card so that it can be processed when needed.

PRINT_SUMMARY

Print the number of each type of card read from the control file.

PRINT_CARDS

Print the cards read from the control file.

GET_STRING

This routine fetches consecutive fields from a card.

PROCESS_name_CARD

Process each "name" card, where name is one of the card types. This involves creating the years range given by the YEARS card, loading data files, etc.

CREATE_PROJECTION_MACROS

Create all the macros of the form "NAME=NAME" where NAME is the name of a scenario in the path starting from a name given in a projection card to the root of the tree defined in the :TREE section of the summary files. The macros define the sections of the data files (RPF and IPF) that are to be read.

4.3.2 REPORT_WRITER

The report writer class produces the result file after processing is finished in the model. Each type of record in the result file has a corresponding routine that writes the record. For example, PRINT_DP_RECORDS prints the disposal operation records (DP,) in the result file. The driver for this class, PRINT_RESULT_FILE, calls each of the print routines, which are self-explanatory.

4.4 SWPM DATA DICTIONARY

The "global" objects used in SWPM include general variables that are needed throughout the code, such as YEAR, and variables included in the operation and waste generator objects. In the following list, variables marked with an asterisk (*) are those that have values supplied by the input data files (RPF and IPF).

The variables that do not belong to any specific frame are described in the following listing.

TYPE	NAME	DESCRIPTION
STRING	*LIB_COMMENT	General comment for projection libraries. It has the COMMENT_PUTDEM method attached.
LIST	LIB_COMMENT_LIST	List for storing the LIB_COMMENT strings.
INT	*COST_UNITS	Value defining the cost units used in all cost calculations. This version of the model assumes that all units are consistent, so this is used only for reporting.
INT	*VOL_UNITS	Units of volume.
LIST	CATEGORY	List of categories given in the SMCF.
LIST	TREATMENT	List of all treatments given in the SMCF.
LIST	ALL_WG	List of all waste generators given in the SMCF.
LIST	COMP_LIST	Comprehensive list including all waste generators, waste classes, and treatments included in the SMCF.
LIST	ABBREV_TYPE	List used for declaring the abbreviation types. The contents of this list are RANK, SHORTNAME, AND LONGNAME.
STRING	*ABBREV [COMP_LIST, ABBREV_TYPE]	Contains the rank, short name, and long name for all names in the analysis.

TYPE	NAME	DESCRIPTION
RANGE	YEARS	Range from the first year in the analysis to last year. This must be all possible years. The report writer can control which years are reported.
REAL	INVENTORY [TREATMENT,YEARS]	Inventory at each operation at each time unit.
REAL	PROCESSED	Amount processed at each operation at each year.
REAL	AMT_RECVD [TREATMENT, ALL_WG,YEARS]	Amount received at each operation from each waste generator at each year.
REAL	TREAT_COST [TREATMENT,YEARS]	Cost of treatment at each operation for each year.
REAL	STOR_COST [TREATMENT,YEARS]	Cost of storage at each operation for each year.

Each waste generator instance in the model contains the following variables. Again, those marked with asterisks have values supplied through the input data files.

TYPE	NAME (WASTE_GEN:>)	DESCRIPTION
REAL	*WG_VOL[CATEGORY,YEARS]	The projected volume of waste for each waste generator by year by waste class by waste generator.
REAL	*WG_DIST_F[CATEGORY,TREATMENT]	The distribution fraction for each waste generator by waste class by treatment.
LIST	WG_WC	List of waste classes given in the SMC for each waste generator.
STRING	*WG_DIST_COM[CATEGORY]	The comment for waste generator distributions by waste class. The COMMENT_PUTDEM method is attached to this object.

TYPE	NAME (WASTE_GEN:>)	DESCRIPTION
LIST	WG_DIST_COM_LIST [CATEGORY]	List for storing the actual distribution comments.
STRING	*WG_DESCRIPTION	The general comment for each waste generator. The COMMENT_PUTDEM method is attached to it.
LIST	WG_DESCRIPTION_LIST	List for storing the actual waste generator description comments.
STRING	*WG_CLASS_OF	"List" of groups to which each waste generator belongs. It is not used in the model but is carried through from the data file for the report writer.
STRING	*WG_CATEGORY_COM[CATEGORY]	The comment for each for each waste class within each waste generator.
LIST	WG_CATEGORY_COM_LIST[CATEGORY] [CATEGORY]	List for storing the actual waste-class comments.

Each operation instance in the model has the following variables. Those marked with an asterisk have values supplied by the input data files.

TYPE	NAME (OPERATION:>)	DESCRIPTION
REAL	*TRT_CAPACITY[YEARS]	Capacity of each operation by year.
STRING	*TRT_WSTCAT	Name of the waste class which this operation handles.
STRING	*TRT_CLASS_OF	"List" of classes to which each operation belongs. It is not used in the model but is carried through from the data file for the report writer.
REAL	*TRT_RATIO	Volume adjustment factor for waste going through each operation.

TYPE	NAME (OPERATION:>)	DESCRIPTION
REAL	*TRT_STOR_RATE	Cost of storage per unit volume at each operation.
REAL	*TRT_TREAT_RATE	Cost of treatment per unit volume at each operation.
STRING	*TRT_DESCRIPTION	General comment for each operation. The COMMENT_PUTDEM method is attached to this object.
LIST	TRT_DESCRIPTION_LIST	List for storing the actual descriptions.
LOGICAL	*TRT_IS_DISPOSAL	Indicates if an operation is a treatment or a disposal. The default value is false unless told otherwise at the member level in the projection file.
STRING	*TRT_CAP_COM	Comment describing the capacity at each operation. The COMMENT_PUTDEM method is attached to this object.
LIST	TRT_CAP_COM_LIST	List for storing the actual capacity comments.
REAL	*TRT_DIST_F[TREATMENT]	The distribution fraction for each operation to other treatments or disposal operations. This is zero for operations that are disposals.
STRING		Comment describing the distribution for each operation.
LIST	TRT_DIST_COM_LIST	List for storing the actual distribution comments.
LIST	TRT_FROM_WG	List of waste generators which contribute to each operation.
LIST	TRT_FROM_OPR	List of treatment operations which contribute to each operation.

TYPE	NAME (OPERATION:>)	DESCRIPTION
LIST	TRT_TO_OPR	List of operations which each operation distributes to (this can be taken directly from the TRT_DIST_F array.
REAL	TRT_AMNT_RECVD[ALL_WG]	Volume received from each waste generator.
REAL	TRT_INVENTORY	Total volume in storage for each operation. This is the source for the actual throughput for each treatment or disposal.
REAL	TRT_PROCESSED	Volume of waste processed by each operation during the current year. This value is always less or equal the TRT_CAPACITY[yr].
LOGICAL	TRT_ACTIVE	True if the operation is active. This is used to trap attempts to have recursion through a treatment.
LOGICAL	TRT_FINISHED	True if the values for the operation have been calculated for the year.
REAL	TRT_STOR_COST	Total cost of storage for the current year.
REAL	TRT_TREAT_COST	Total cost of treatment for the current year.

5.0 OUTPUT REPORT WRITER

This chapter describes the structure and functions of the ORW, which was developed to provide standard reports of results generated by the SWPM model. The ORW is implemented using ADL, with support from the ADL intrinsic function "ARRAY_TOTAL."

5.1 ORW OVERVIEW

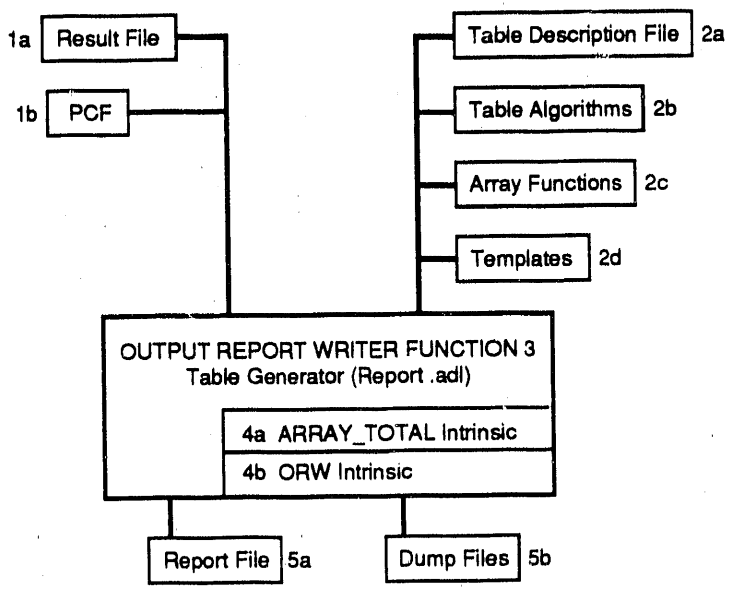
The processing flow of the ORW software is the following:

- The SWPM User Interface invokes ORW by executing the ADL translater with the main ORW program (report.adl).
- The PCF is loaded.
- The results from the SWPM are loaded, and ADL objects needed by ORW are created.
- Each of the reports listed in the PCF is created and written to the report file.
- The report is sent to the printer if a printer device is given in the PCF.

The data inputs to ORW are results generated by the model and a printer control file describing the tables to generate. Other inputs to ORW are in the form of control or configuration files. The "OUTPUT REPORT WRITER FUNCTION" processes these inputs and outputs a report file and an optional dump file containing ORW debug information. Figure 5.1 provides and overview of the ORW data flow.

5.2 FUNCTION DESCRIPTIONS

The ORW is composed of ADL functions and data files. The following table summarizes the files used by ORW. Included with each file is a short description of its contents.



1a - The results from the SWPM run. The data in the result file is used to create tables.

1b - The PCF specifies the tables to generate, where to write the report file, the source of model results, and the data to use in generated tables.

2a - The table description file is used to specify how tables are generated.

2b - Table algorithms written in ADL that set up table data for each table.

2c - Array functions are used to process rows and columns of data for each table. The array functions may compute aggregates or percentages given the specification for a table.

2d - Template files provide format rules for tables created by ORW. The format of report pages and tables can be described with template files.

3 - The "OUTPUT REPORT WRITER FUNCTION" (or report.adl) is an ADL main program which prompts for a printer control file (PCF). After receiving a PCF the report writer function generates each of the tables listed in the PCF.

4a - The ARRAY_TOTAL intrinsic is used by report.adl to insert column or row totals into a data matrix for a table. ARRAY_TOTAL additionally updates or creates a list of row and column labels for the table.

4b - The ORW intrinsic is used by report.adl to create templates for the formating of tables and report pages.

5a - The report file is created by report.adl from data in the result file. The file contains all tables specified in the PCF.

5b - An optional output file created to trace the ADL portions of ORW.

FIGURE 5.1. Overview of the ORW Data File

FILENAME	DESCRIPTION
REPORT.ADL	Contains the main ORW program written in ADL.
TABLE.ALG	The table generation algorithm used by ORW to create tables for SWPM.
LINE.ALG	Algorithms used by table algorithms for the processing of rows of table data.
LOAD_PCF.FUN	ADL code that reads the PCF.
LOAD_DAT.FUN	ADL code which reads the result file.
UTILITY.FUN	ADL utility functions used to manipulate ADL lists.
HEADER FUN	Outputs a standard header for tables generated for SWPM.
ERROR.FUN	Contains routines used to trace the ADL portion of ORW.
TBL_DESC.DAT	The data file containing the description for all tables generated for SWPM.
ALL.PCF	A printer control file used to generate all the possible SWPM tables.

The following sections describe each of the above functions in detail.

5.2.1 MAIN PROGRAM (REPORT.ADL)

The main ADL program (report.adl) prompts for a PCF and then generates all the reports listed in the PCF. The following actions occur when report.adl is used.

- Prompt for a printer control file (PCF). In the case of SWPM the user interface provides a PCF (SWPM.PCF) when ORW is invoked.
- Load the PCF into the ORW data structures.
- Load the results of the a SWPM model run. The name of the result file is specified in the PCF.
- Create and output all tables listed in the PCF. The ADL intrinsic function ARRAY_TOTAL is used to calculate row and column totals; while intrinsic ORW is used to format tables.

- Output the tables to a printer if a printer device is given in the PCF.

5.2.2 TABLE GENERATION (TABLE.ALG)

Table generation includes the formatting of data comprising a table and the building of row and column lists that are used for row/column headings. Tables are generated for each table that the user selects (iterative calls to the TBO function from report.adl). The functions of the ADL function TBO are described below.

- Build the row and column heading lists specified by "ROW_n," or "COL_n" in TBL_DESC.dat (table description file) for the table that is being created.
- For each row heading built above, call a line algorithm (line.alg) that selects row data for the table.
- For each column heading, output the row built above to a data matrix.
- Return the data matrix (the data for the table) and the row and column headings to report.adl.

5.2.3 SELECTING DATA FOR TABLES (LINE.ALG)

The line.alg routines select data for each row in the table. Each row returned by routines in line.alg are return to the TBO function, which builds a complete table of data (See LINE.ALG).

5.2.4 LOADING PRINTER CONTROL FILE (LOAD_PCF.FUN)

This function is called from report.adl to read the PCF. Data in the PCF is used in controlling which reports are generated.

5.2.5 LOADING THE RESULT FILE (LOAD_DAT.FUN)

The result file is loaded by functions in LOAD_DAT.FUN. These functions build lists of operations, WG (waste generators), WC (waste classes), and others which are used as a basis for the contents in table rows and columns (See LOAD_DAT.FUN).

5.2.6 ORW UTILITY FUNCTIONS (UTILITY.FUN)

This is a general purpose ADL routines used by ORW. Routines included in UTILITY.FUN are used to create text lists for routine TBO().

5.2.7 OUTPUTTING TABLE HEADERS (HEADER.FUN)

The header routine is called from report.adl when "Header" is encountered in a table list (i.e., contained in the PCF). The header is printed at the top of each report. It includes the following.

- Case description (includes case name, and author)
- Model run date and report date
- All input files used by the model
- WASTE GENERATORS used by the modelTREATMENTS used by the model
- DISPOSAL OPTIONS
- SWPM INPUT DECK (name of the input deck used for this SWPM run).

5.3 FILES USED BY ORW

The ORW uses the printer control file (PCF), discussed in Chapter 6.0, and the Table Description File (TBL_DESC.DAT) to produce formatted reports.

The table description file contains three association lists containing the information required to create the table/report, and a list containing the title information for the table. The first list contains the instructions required to calculate the numeric data and labels for the array. The second list is an association list containing the information regarding totals, subtotals, and stripping of null rows/columns from the array. The third list contains an association list of information needed to format the table for printing.

TEXT_LIST

TITLE_n_m = "string"	where	"n" is the title label. "m" is the subtitle label. "string" is the text to be inserted in the title.

	example	TITLE_1_1 = "Table 1.1a "
		TITLE_1_2 = "Aggregate Receipts"

ARRAY_LIST

FUNC = "string"	where	"string" is the name of the main function used to calculate the table contents.
	example	FUNC = "TBO";

ROW_n = "string"
COL_n = "string"
	where	"n" is the class level for the row label.
		"string" is the data type assigned to the level or a list of labels.

	example	ROW_1 = "WG";
		ROW_2 = "PH2,ENGNR,OPERT,DECOM";
		COL_1 = "YEAR".

ALG_n = "level,alg_name"	where	"n" is a number used to make the label unique. "level" is the level to apply the algorithm, this may be a type, a label, or the word "ALL". "alg_name" is the name of an algorithm included in the run.

	examples	ALG_1 = "ALL,REF_FUNC";
		ALG_2 = "ENGNR,ENGNR_COST";
		ALG_3 = "WG,RECEIPT_WG".

The table description field, which is loaded by the GET_CMD function, is used to describe how each table is generated.

The GET_CMD function reads the table description file saving control information about each table. The table control information includes the table's heading, row/column heading (for each row or column level), the ADL array that contains the data for the table, the line algorithm to use for selecting table rows, and the row or column totals to generate.

6.0 SWPM FILES

This chapter explains the function of the files used in the SWPM software to communicate between model modules and to load input data transferred from the SWPM database. The following extensions specify the file type.

• RPF	Reference Projection File
• IPF	Incremental Projection File
• RPS	Reference Projection Summary
• IPS	Incremental Projection Summary
• SMC	Model Control File for SWPM
• RSL	Result File produced by SWPM
• PCF	Print Control File
• CAS	Case description save file for User Interface (UI)
• DD	Data Dictionary
• REP	Final reports
• UPF	Username/Password File
• PCD	Print Control Data File
• OUT	Redirected Standard Output File.

		RPF	IPF	RPS	IPS	SMC	RSL	PCF	CAS	DD	REP	UPF	PCD	OUT
UI	I			X	X				X			X	X	
	O					X		X	X					
SWPM	I	X	X	X	X	X				X				
	O						X							X
ORW	I						X	X						
	O										X			X
PFI	I	X	X	X	X									
	O		X		X									X

6.1 REFERENCE PROJECTION FILE (RPF)

The source of all original knowledge for the SWPM comes from this file. It is produced by the SWPM database and contains all the values used by the model, such as waste volumes, distribution fractions, storage costs, etc. The file must be a valid ADL data file.

The order of the data statements is critical because all the data are loaded at the member frame level. Through judicious use of the #ifdef directive, 'inheritance' is implemented. If the macro portion (scenario name) of an #ifdef statement has been defined, the data statement within the #ifdef directive is read and overloads any previous data. Thus, each scenario provides a delta to the existing data, and any object can have its information provided in different paths (tree structure) through #ifdef statements.

For example, the PNL waste generator may consist of a user projection followed by several incremental pieces of data. This "tree" can have several branches in it as is illustrated in Figure 6.1.

An analyst may decide to use any one of the "boxes" as the starting point for the PNL waste generator. If the analyst picks PNL_ALT1, the order of the data loading must be PNL_UP, PNL_MOD1, PNL_ALT1. The parent/child relationship information is contained in the :TREE section of the RPS file.

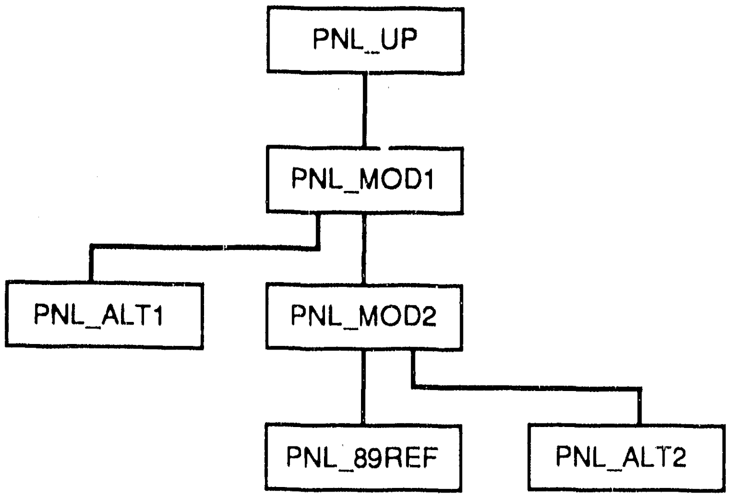


FIGURE 6.1. Reference Projection Tree

The key to using this solution is the RPF. The RPF must provide the information for each entity in a top-down fashion and surround each item with the scanner directives #IFDEF and #ENDIF. For example, the situation described in Figure 6.1 would be represented in the RPF with the following statements.

```
#IFDEF PNL_UP
     ! data from the PNL_UP projection
     PNL:WG_VOL[ ...
     PNL:WG_DESCRIPTION = " ...
     ...
#ENDIF
#IFDEF PNL_MOD1
     ! data from the PNL_MOD1 projection
     PNL:WG_VOL[ ...
     PNL:WG_DESCRIPTION = " ...
     ...
#ENDIF
#IFDEF PNL_ALT1
     ! data from the PNL_ALT1 projection
     PNL:WG_VOL[ ...
     PNL:WG_DESCRIPTION = " ...
     ...
#ENDIF
#IFDEF PNL_MOD2
     ! data from the PNL_MOD2 projection
     ...
#ENDIF
#IFDEF PNL_89REF
     ! data from the PNL_89REF projection
     ...
#ENDIF
#IFDEF PNL_ALT2
     ! data from the PNL_ALT2 projection
     ...
#ENDIF
```

The order of multiple planning does not matter as long as any path from the root of the tree (PNL_UP) to any leaf (PNL_ALT1, PNL_89REF, or PNL_ALT2) is in descending order. Thus, the #IFDEF section for PNL_ALT1 could have come any place after its position as given above.

The following "template" file gives the format of the projection file.

```
HEADER
! declare the ranges, and lists for loaddata
     ALL_WG = (wg1, wg2, ...);
     TREATMENT = (operation1, operation2, ...);
     CATEGORY = (wasteclass1, wasteclass2, ...);
     COMP_LIST = (ALL_WG,TREATMENT,CATEGORY);
     abbrev_type = (RANK,SHORTNAME,LONGNAME);
     years = start..end;

! declare the abbreviation array.  It is used to give the rank, the short
  name, and the long name of each name used in the analysis.
     STRING ABBREV[COMP_LIST,abbrev_type];

! declare the library comment, though it is not currently used.
     STRING LIB_COMMENT;

! the units of cost and volume must be consistent throughout the file.
  The  COST_UNITS and VOL_UNITS indicate the magnitude of each cost and
  volume.  If COST_UNITS is 1, then it implies that all costs are given
  in dollars.  If VOL_UNITS is 1000, then it implies that all volumes are
  given in thousands of cubic feet.
     INT COST_UNITS;
     INT VOL_UNITS;

! Declare the waste generator information objects.
     REAL wg1:WG_VOL[CATEGORY,years];
     REAL wg2:WG_VOL[CATEGORY,years];
     ...

     REAL wg1:WG_DIST_F[CATEGORY, TREATMENT];
     REAL wg2:WG_DIST_F[CATEGORY, TREATMENT];
     ...

     STRING wg1:WG_DIST_COM[CATEGORY];
     STRING wg1:WG_DIST_COM[CATEGORY];
     ...

     STRING wg1:WG_DESCRIPTION;
     STRING wg2:WG_DESCRIPTION;
     ...

     STRING wg1:WG_CLASS_OF;
     STRING wg2:WG_CLASS_OF;
     ...

     STRING wg1:WG_CATEGORY_COM[CATEGORY];
     STRING wg2:WG_CATEGORY_COM[CATEGORY];
     ...
```

```
! declare the treatment information objects

    !define which waste category this treatment handles
    STRING trt1:TRT_WSTCAT;
    STRING trt2:TRT_WSTCAT;
    ...

    STRING trt1:TRT_CLASS_OF;
    STRING trt2:TRT_CLASS_OF;
    ...

    REAL trt1:TRT_RATIO;
    ...

    ! the cost/unit of the implied "storage before treatment" for this
    treatment
    REAL trt1:TRT_STOR_RATE;
    ...

    STRING trt1:TRT_DESCRIPTION;
    ...

    REAL trt1:TRT_CAPACITY[years];

    ! declare capacity comment
    STRING trt1:TRT_CAP_COM;
    ...

    ! cost on treatment throughput per unit of waste
    REAL trt1:TRT_TREAT_RATE;
    ...

    REAL trt1:TRT_DIST_F[TREATMENT];
    ...

    STRING trt1:TRT_DIST_COM;
    ...

    ! Declare the logical for telling if a treatment is a disposal.
    INT trt1:TRT_IS_DISPOSAL;
    ...

END_HEADER
DATA
!  The data section will be normal ADL data file statements with #ifdef
directives around the data.
END_DATA
```

6.2 REFERENCE PROJECTION SUMMARY (RPS)

This file is a summary of the RPF; it contains lists of the objects and associated scenarios that are in the RPF. The different sections of the summary file are delimited by a keyword preceded by a colon. The sections do not have to appear in any specific order and may be repeated. They consist of the following.

• :TREE	Each record in this section is of the form: scenario1 [,scenario2], where scenario1 is a child of scenario2.
• :CLASSES	Each record in this section gives the name of a waste class.
• :SCENARIO	This section gives all waste generator, operation, and treatment objects and their associated scenarios. Each record is of the form: TYPE, object name, scenario name, "short description," where TYPE is one of WG, OP, or TRT to indicate if the object name is a waste generator, operation, or treatment, respectively.
• :OPTYPE	This section lists the names of available operation types. These types are concatenated with waste class names to create operation names.

6.3 INCREMENTAL PROJECTION FILE (IPF)

The IPF file contains "changes" to the data stored in the RPF and new data. When the user modifies existing data or creates new objects through the User Interface, that data are stored in the IPF. It follows essentially the same format as the RPF except for some #mode statements, which guide the PFI in updating the file.

The purpose of the dual #mode statements is to allow the PFI to know when to write out a mode statement when copying the existing data section to the new IPF. The scanner does not signify when a #mode statement is scanned; therefore, the PFI must be told when one is scanned through the dual statements.

The header section of the IPF is always in the same order so that the PFI can read it easier. The order is as follows.

```
HEADER
     OPERATION = (list of operations);
     WC = (list of waste classes);
     WG = (list of waste generators);
     YEARS = lowyr..highyr;
     COMP_LIST = (OPERATION,WC,WG);
     ABBREV_TYPE = (RANK,SHORTNAME,LONGNAME);

     object declarations ...
END_HEADER
```

The data section contains the dual #mode statements. When the PFI initializes the SCANNER module, it defines the macro PFI_MODE. Thus, when the scanner returns the token "PFI_MARKER" to the PFI, the PFI knows that it is in the middle of one of the dual #mode statements and can reproduce it when copying the existing data section to the updated file. When the LOADDATA module is reading the IPF, it does not have PFI_MODE defined and sees only the data statements following defined scenario names (i.e., they act exactly like the #ifdef statements in the RPF). The format of the data section of the IPF is as follows.

```
DATA
#MODE PFI_MODE
     PFI_MARKER scenario1
#MODE PFI_MODE, scenario1
     data statements ...

#MODE PFI_MODE
     PFI_MARKER scenario2
#MODE PFI_MODE, scenario2
     data statements ...

...
#MODE LOADDATA, PFI_MODE
END_DATA
```

6.4 INCREMENTAL PROJECTION SUMMARY (IPS)

The IPS file summarizes the IPF. Its format is the same as the RPS.

6.5 SWPM CONTROL (SMC)

This file specifies options and files that are to be used in an analysis, including the file names of the data dictionary and data files. Also, a result file can be specified, otherwise the results are written to standard output. The SMC file is not an "ADL" file; it is read by ADL code and, thus, does not support the usual directives such as #DEFINE, #IFDEF, #MODE, etc. Each field in a record is delimited by a comma. The following records are supported.

• DATAFILE	file-spec [,LOGSKIP]
• SUMRYFILE	file-spec
• DATADICT	file-spec
• RESULT	file-spec

The DATAFILE record gives the name of a RPF or an IPF, and the LOGSKIP option causes messages to be issued when data is skipped. The DATADICT record specifies a data dictionary to load. The SUMRYFILE contains a "tree" section describing the scenario tree for all waste generators and operations. Any number of DATAFILE, DATADICT, and SUMRYFILE records can be specified and will be loaded in the order given. The RESULT record gives the name of the file to write the final values to. This is the file that the Report Writer Task will use to generate output tables.

Options that are supported in the SMC are name of person specifying run, Case ID, and comment for the run, waste generators, waste classes, operations, and associated projections (scenario names). These are:

• AUTHOR	"name of person"
• CASEID	"alphanumeric name"
• CASECOMMENT	"single line comment describing case"
• YEARS	first_year_of_analysis, last_year_of_analysis
• WG	"waste generator" [, "waste class," "waste class," ...]
• TREATMENT	"operation," "operation," ...

- PROJECTIONS "projection name," "projection name," ...

Because a record is, by definition, a single line of input, each group or list of items must be on a single line. If it does not fit, it can be continued on a second record/line by using the same key word (this applies to records that support a list of items such as the TREATMENT record).

The PROJECTIONS record gives all projections (scenarios) for a case, including those for waste generators and operations. The SWPM uses the SUMRYFILEs to get the tree structure of those projections. An SMC file might look like the following.

```
! comments are supported with a ! in the first column
! SAMPLE SMC
AUTHOR, John J. Johnson
CASEID, CASE_5
CASECOMMENT, Describe the case here.  You get only one line.
! define the time period to report on
YEARS, 1992, 2005
! specify data dictionary
DATADICT, e:\swpm\dd\swpm6.dd
! specify Reference Projection file and its summary file
DATAFILE, e:\swpm\baselib\swpm6.rpf
SUMRYFILE, e:\swpm\baselib\swpm6.rps
! specify Incremental Projection file to change some data in Reference
!    Projection file and specify its summary file
DATAFILE, swpm6.ipf, logskip
SUMRYFILE, swpm6,ips
! the result file
RESULT, testcase.rsl
PROJECTIONS, be_up, be_modl, lb_up, whc_2345_up, ...
! specify the waste generators with appropriate waste classes
WG, AMES, CH_LLW_LOW
PROJECTIONS, AMES_FD89
WG, ARGONNE, CH_LLMW_LOW, CH_LLW_LOW, CH_TRU
WG, ARGONNE, CH_TRUM, RH_LLMW_LOW, RH_LLW_INT
WG, ARGONNE, RH_LLW_LOW, RH_TRU
PROJECTIONS, ARGONNE_FD89
! specify the operations with their projections
TREATMENT, ASH_IMMOB_CH_LLMW_LOW
PROJECTIONS, ASH_IMMOB_CH_LLMW_LOW_PRELIM
TREATMENT, CAN_RH_LLW_GTCH
PROJECTIONS, CAN_RH_LLW_GTCH_PRELIM
```

6.6 DATA DICTIONARY (DD)

The data dictionary file is used for declaring the global and frame variables used in the ADL SWPM. The file must be a valid data dictionary file as described in the ADL User's Guide. Variables declared in the SWPM data dictionary include global variables such as COST_UNITS, VOL_UNITS, INVENTORY, STOR_COST, etc. Also included are variables for the operation (TRT_) and waste generator (WG_) frames.

Because the indices of some variables are not known until run time, those are not declared in the data dictionary file. These include some of the waste generator frame variables that are dimensioned by waste class (WG_VOL, WG_DIST_F, WG_DIST_COM, WG_DIST_COM_LIST, WG_CATEGORY_COM, and WG_CATEGORY_COM_LIST).

6.7 RESULT FILE (RSL)

When the SWPM runs, it produces a result file containing all the pertinent data from the case. The ORW uses this as its data file for producing reports.

To keep the result file somewhat compact, each record is written in a shorthand notation. The record begins with a short code followed by the data. Some of the records, which have values by year, cover several lines. The following record types are produced.

- TD1 Time and date.
- TD2 ADL and SWPM version information.
- CI Case ID. This should be a short identifier.
- VU Volume units. This is echoed through from the RPF.
- CU Cost units. This is also echoed through from the RPF.
- H Mapping of operation to the waste class it handles.
- DP Disposal operation with rank, shortname, longname, and list of classes to which the treatment belongs.
- TR Treatment operation with rank, shortname, longname, and list of classes to which the disposal belongs.

• WG	Waste generator with rank, shortname, longname, and list of classes to which a waste generator or treatment belongs.
• WC	Waste class with rank, shortname, and longname.
• AR	Amount of waste received at each treatment from each waste generator at each time unit.
• TT	Volume throughput for each treatment at each time unit.
• TC	Capacity of each treatment at each time unit.
• I	Inventory at each treatment's storage at end of each time unit.
• DW	Distribution values for waste generators.
• DT	Distribution values for treatments.
• OV	Operation values. This includes unit cost for storage, unit cost for processing, and the process volume change (applicable only for treatment operations).
• CS	Cost of storage in front of each treatment at each time unit.
• CT	Cost of treatment at each treatment at each time unit.
• C	Comments. These include the comments from libraries, waste generator descriptions, distribution descriptions, etc. The comment portion will be enclosed in double quotes. Put-demons on the comment variables used in the model will concatenate successive comments for the same variable together with a line feed between each one.
• F	File names (with date) of all files used in the analysis.
• MF	Echo of records from the SMC file.

6.8 PRINTER CONTROL FILE (PCF)

The PCF is loaded from the routine LOAD_PCF (LOAD_PCF.FUN). It is created from the User Interface when the user selects the desired tables. The PCF is used to command the report writer to generate a set of reports. The main report program "report.adl" prompts for the PCF filename. All PCF file records have the following format:

IDENT = VALUE

where: IDENT is the data identifier or tag for the data. Another way to look at it as the name of an association.

VALUE is the data "associated" with ident, or the value of the association.

For example "Date=11:00:00 11/19/1989."

The following INDENT types (or record types) are supported in the PCF.

"Date="	Time and Date of request values have the form HH:MM:SS MM/DD/YYYY. For example "Date = 10:22:33 11//19/1989".
"Resf="	Path and File Name of result file. The filename (or value) must conform the filename and pathname conventions supported by the host operating system. For example: "Resf=d:\orw\test.pcf".
"Repf="	Path and File Name of report output file. The filename (or value) must conform to the filename and pathname conventions supported by the host operating system. For example: "Repf=d:\orw\test.rep".
"Tids="	Table Identifier for the list of tables to be generated. Multiple tables can be listed either by using Tids multiple times or by listing tables, separated by commas, with one Tids. For example: "Tids=1.1a," "Tids=1.2a" or "Tids=1.1a, 1.2a". NOTE: A header can be printed by including a "Tids=Header" record when a header is appropriate.
"Ptyp="	The printer type.
"Pdev="	Print Device.
"Lids="	Label identifier used to specify the valid values for each object used in a report. Only values specified in Lids records are used in reports. For example: "Lids=WG/AMES_LAB PNL" defines that the only valid WG (waste generators) in a report are AMES_LAB and PNL. Data associated with other waste generators is not printed. Lids is also used to control the order of data in tables. In the last example AMES_LAB data would be printed before PNL data.

All of the above record types can be in the PCF, and may be in any order. Other identifiers are not allowed. An example PCF follows.

```
Date =  10:22:33 07/11/1989
File =  d:\swpm\report\myreport.dat
Tids = 1.1a
Tids = 1.2c
Tids = 6.1a
ptyp = HPlaserjet
pdev= lpt1
lids = WG\PNL, AMES_LAB
```

6.9 PRINTER CONTROL DATA FILE (PCD)

The contents of this file (SWPM.PCD) are displayed for the user by the User Interface to allow the user to select the tables to be produced by the ORW. The user's selections are used to create a new version of SWPM.PCF. The contents of SWPM.PCD are as follows:

```
Header prints all comment and run information
-
1.1a Receipt by waste generators and waste classes (vol)
1.1b Receipt by waste generator categories and waste classes (vol)
1.1c Receipt by waste classes (vol)
1.2a Receipt by waste classes and waste generators (vol)
1.2b Receipt by waste class categories and waste generators (vol)
-
2.1a Storage volume by treatment (vol)
2.1b Storage volume by treatment category and treatment (vol)
2.2a Cost of storage by treatment ($)
2.2b Cost of storage by treatment category and treatment ($)
2.3a Annual incremental storage volume by treatment (vol)
2.3b Annual incremental storage volume by treatment category and
     treatment (vol)
-
3.1a Treatment throughput volume by treatment (vol)
3.1b Treatment throughput volume by treatment category and treatment
     (vol)
3.2a Cost of treatment ($)
3.2b Cost of treatment by category and treatment ($)
3.3a Treatment utilization by treatment (%)
3.3b Treatment utilization by category and treatment (%)
-
4.1a Storage volume by disposal option (vol)
4.1b Storage volume by category and disposal option (vol)
4.2a Cost of storage by disposal option ($)
4.2b Cost of storage by category and disposal option ($)
```

4.3a Incremental change in storage before disposal (vol): by disposal option
4.3b Incremental change in storage before disposal (vol): by disposal category and option
-
5.1a Disposal volume (vol): by disposal option
5.1b Disposal volume (vol): by disposal category and option
5.2a Cost of disposal ($): by disposal option
5.2b Cost of disposal ($): by disposal category and option
-
6.1a Waste distribution fractions (%): by waste class and waste generators
6.1b Waste distribution fractions (%): by treatment/ disposal option
6.2a Operation description table
6.2b Operation capacities summary (vol)

6.10 CASE LOG FILE (CAS)

The User Interface maintains all necessary information to recreate a SWPM run in a CAS file unique to that case. The CAS file contains a header whose records define general case information, such as the case's name, author, name of the case's parent case, and the RPF, RPS, IPF, IPS and DD files used by the case. Following the header is a sequence of records that define the SWPM data objects that the user preselected for the case. Waste classes, waste generators, treatments, and disposals are defined on WC, WG, TRT and DP records, respectively. These records include the name of the object and the name of the scenario that defined it for the subject case. Thus, the CAS file contains all information necessary to reload a case's data from the RPF and IPF files. A short sample of a CAS file is shown below:

```
ID = SWPM User Interface -- Version 1.0  (01/17/90)
VERNTEST
VERN_NEW
desc
vern crow
vern crow
05.RPF
05.RPS
05.IPF
05.IPS
SWPM.DD
VERNTEST.RSL
1990
2020
```

```
WC, CH_HAZ
WC, CH_LLMW_GTCH
WC, CH_LLMW_HIGH
WC, CH_LLMW_INT
WC, CH_LLMW_LOW
WC, CH_LLW_GTCH
WC, CH_LLW_HIGH
WC, CH_LLW_INT
WC, CH_LLW_LOW
WC, CH_TRU
WC, CH_TRUM
WG, WHC_221T, WHC_221T_FD89
WG, WHC_308, WHC_308_FD89
WG, WHC_TWRT, WHC_TWRT_FD89
WG, BECHTEL, BECHTEL_VERN_TEST
TRT, BIOLOGICAL_CH_TRU, BIOLOGICAL_CH_TRU_PRELIM
DP, BOX_CH_LLW_INT, BOX_CH_LLW_INT_PRELIM
DP, BOX_CH_LLW_LOW, BOX_CH_LLW_LOW_VERN_NEW
TRT, COMB_COMP_CH_LLW_INT, COMB_COMP_CH_LLW_INT_PRELIM
DP, DRUM_CH_HAZ, DRUM_CH_HAZ_PRELIM
DP, DRUM_CH_LLMW_LOW, DRUM_CH_LLMW_LOW_PRELIM
DP, DRUM_CH_LLW_LOW, DRUM_CH_LLW_LOW_PRELIM
DP, DRUM_CH_TRU, DRUM_CH_TRU_PRELIM
DP, DUMP_TRUCK_CH_LLW_LOW, DUMP_TRUCK_CH_LLW_LOW_PRELIM
TRT, METAL_GTH_CH_TRU, METAL_GTH_CH_TRU_PRELIM
TRT, METAL_LTH_CH_HAZ, METAL_LTH_CH_HAZ_PRELIM
TRT, METAL_LTH_CH_LLW_INT, METAL_LTH_CH_LLW_INT_PRELIM
TRT, METAL_LTH_CH_LLW_LOW, METAL_LTH_CH_LLW_LOW_PRELIM
TRT, METAL_LTH_CH_TRU, METAL_LTH_CH_TRU_PRELIM
DP, OTHER_2_CH_LLW_INT, OTHER_2_CH_LLW_INT_PRELIM
DP, BOX_CH_TRU, BOX_CH_TRU_VERN_TEST
```

6.11 USERNAME/PASSWORK FILE (UPF)

The file SWPM.UPF contains the list of users authorized to use the SWPM User Interface. It includes their passwords and an indication of their status. A short sample UPF follows:

```
Joe Doaks,pass1
Polly Privileged,pass2,priv
```

In the above example, the second user will receive special treatment when logged in to the SWPM User Interface.

6.12 REPORT FILE (REP)

REP files are produced by the ORW in a format suitable for printing. The User Interface facilitates printing these files on an HP Laserjet printer by invoking the command procedure ORWLASER.CMD in response to the user's selection of the "Print" menu item.

6.13 REDIRECTED STANDARD OUTPUT FILE (OUT)

The User Interface redirects standard output to ADLSTD.OUT so that standard messages from the PFI routines, the SWPM, and the ORW will be caught in a file for post-mortem analysis. Normally under PM, standard output is discarded.

6.14 DIRECTORY STRUCTURE

When reading and writing files, SWPM expects a specific set of directories to exist. These directories also must match a defined structure. The following list shows all of the required hard disk directories, which contain the electronic files composing the SWPM system:

E:\ADL\ADL.EXE	The ADL Interpreter, an OS/2 Protected Mode program
E:\ADL\SWPMOD.ADL	The source code of the SWPM model
E:\ADL\REPORT.ADL	The source code of the Report Generator
E:\ADL\MODELS.H	Source code included by the SWPM model
E:\ADL\CMPNENTS.H	Source code included by the SWPM model
E:\ADL\REPWRIT.H	Source code included by the SWPM model
E:\ADL\SMC.H	Source code included by the SWPM model
E:\ADL\METHOD.H	Source code included by the SWPM model
E:\ADL\EPU_MSG.DAT	Control file used by ADL.EXE
E:\ADL\ADLDECOD.DAT	Control file used by ADL.EXE
E:\ORW	Directory full of Report Generator support files
E:\WHC\SWPM.EXE	The User Interface, an OS/2 Protected Mode program
E:\WHC\ORWLASER.CMD	Laser printer command file for the User Interface
E:\WHC\SWPMPCF.DAT	Data file used by the User Interface
E:\WHC\SWPM.UPF	Username and password file for the User Interface

In addition, the E:\WHC director contains any SWPM input and output files for test and production cases (e.g., *.DD, *.RPF, *.RPS, *.IPF, *.IPS).

END

DATE FILMED

01 / 29 / 91